Agriculture
Canada

Research Branch
Direction générale de la recherche

Technical Bulletin 1989-4E

BUGWATCH

Program reference manual

Canada

BUGWATCH

Program reference manual

JOHN M. YEE and S. YEE
Research Station
Vineland, Ontario

Technical Bulletin 1989-4E

Research Branch
Agriculture Canada
1989

Copies of this publication are available from
Director
Vineland Research Station
Research Branch, Agriculture Canada
Vineland, Ontario
L0R 2E0

Produced by Research Program Service

©Minister of Supply and Services Canada 1989
Cat. No. A54-8/1989-4E
ISBN 0-662-16764-3

Acknowledgments
The Ontario Ministry of Agriculture and Food provided financial support to S. Yee

Cover illustration
The dots on the map represent Agriculture Canada research establishments.

TABLE OF CONTENTS

Section Page

1. INTRODUCTION 1

2. PROCEDURES AND FUNCTIONS 3

3. FILES 8

4. PROCEDURE - DATAFILE MATRIX 24

REFERENCES 27

APPENDIX 1. DEVELOPMENT RATE FUNCTION ...... APPENDIX 1 - 1

APPENDIX 2. LISTING OF DATAFILES IN SUBDIRECTORY STUPFILE .... APPENDIX 2 - 1

APPENDIX 3. BUGWATCH PROGRAM FILES

BUGWATCH.PAS .... APPENDIX 3 - 1
STR_PROC.PAS .... APPENDIX 3 - 11
SITEPROC.PAS .... APPENDIX 3 - 14
DATEPROC.PAS .... APPENDIX 3 - 24
INP-WTHR.PAS .... APPENDIX 3 - 30
INP-TRAP.PAS .... APPENDIX 3 - 37
CALCULAT.PAS .... APPENDIX 3 - 46
PRINTREP.PAS .... APPENDIX 3 - 64
REGIONAL.PAS .... APPENDIX 3 - 79
GRAFTRAP.PAS .... APPENDIX 3 - 91

SUMMARY

BUGWATCH is an IBM-PC based pest monitoring tool designed primarily for apple pests. It assists the Ontario Ministry of Agriculture and Food (OMAF) pest management specialists in analyzing the effect of weather on pest development and in summarizing trap count field data.

This document presents the following:

(1) disk directory tree and subdirectory contents
(2) description of all the data files
(3) procedure/data file access matrix, and
(4) a complete program source code listing in an appendix.

RÉSUMÉ

BUGWATCH est un programme qui peut être utilisé avec l'ordinateur PC d'IBM et sert principalement à la surveillance de la lutte contre les ravageurs des pommiers. Les spécialistes de la lutte contre les ravageurs du ministère de l'Agriculture et de l'Alimentation de l'Ontario y ont recours pour analyser l'effet des conditions atmosphériques sur le développement des ravageurs et pour établir un sommaire du nombre d'insectes recueillis dans les pièges.

Le présent document contient les renseignements suivants :

1) arbre du répertoire de disquettes et contenu du sous-répertoire,
2) description de tous les fichiers de données,
3) méthodes et matrice d'accès aux fichiers de données, et
4) liste complète des codes sources des programmes en annexe.

1. INTRODUCTION

The BUGWATCH system is an IBM-PC based pest monitoring tool designed primarily for apple pests. It assists the Ontario Ministry of Agriculture and Food (OMAF) pest management specialists in analyzing the effect of weather on pest development and in summarizing trap count field data.

The program evolved from a previous one developed at Vineland Research Station (Trottier [1977], Koziol [1978], Hagley et al. [1980]); however, development is not based on linear heat units, but on nonlinear developmental units (Harcourt and Yee [1982]).

The program was written in Turbo Pascal version 3.0 (Borland [1985]) on an IBM-PC/XT operating under PC-DOS 2.10. The machine has 640K main memory, a math coprocessor, and a colour graphics adapter for screen graphics. An Epson printer is the hardcopy output device.

BUGWATCH processes the following input data:
i) daily maximum and minimum temperatures in decimal Celsius degrees,
ii) trap catch data for various types of traps for 6 pests.
OMAF staff collect weather data from the beginning of March to the end of September. The user enters whether the weather data are actual, predicted, historical or normal. They also record trap catch data. In 1988 there were approximately 40 weather sites and 85 trap catch sites; some sites served as both weather and trap data monitoring sites.

BUGWATCH produces the following output information for each site and each insect:
i) insect lifestage and percent development,
ii) five day forecast of lifestage and percent development,
iii) regional reports of the insect lifestage development,
iv) table of trap catches,
v) table of the insect population events,
vi) graphs of the trap catches - average versus day.
Typically, OMAF staff produce weekly reports for items i) to iii) and end of season reports for items iv) to vi). They may also produce reports for item vi) throughout the season.

The users have little experience using a computer so the system must be simple to use. They want to enter data at any time and in any order they choose, and they want a simple way of correcting input errors.

We made the following design decisions to try to maximize transportability among different computers.

(1) All data are stored in ASCII files.
(2) The program avoids IBM-PC dependent features as much as possible. Most of the IBM-PC dependent program code is located in the graphics routines.
(3) The data entry procedures use a line oriented method rather than a full screen method.
(4) The program uses Turbo Pascal and no other utility packages.

The line oriented, data entry prompting method makes the system easier to use than a full screen data entry method. This method also makes the system more resistant to user misunderstandings.

The BUGWATCH system resides on the hard disk because a floppy disk has insufficient storage space. A site subdirectory will require at least 2K for each data file to accommodate the file directory header. Thus a typical site needs about 20K of disk space not including the actual data.

This document contains all the information and file listings needed to construct a BUGWATCH system. Appendix 2 contains listings of the data files. Appendix 3 contains listings of the BUGWATCH program source code files.

2. PROCEDURES AND FUNCTIONS

The BUGWATCH program is divided into 6 main procedures according to the function performed. These are as follows:

i) input weather
ii) input trap
iii) calculate
iv) print reports (site by site)
v) print regional reports
vi) graphs (site by site)

These procedures access data files stored in monitoring site subdirectories. These data files contain weather data, pest lifestage development data and pest trap catch data.

2.1. PROCEDURE LIST

This section lists all of the BUGWATCH procedures and functions in the order in which they are declared. Procedures and functions declared locally within another are indicated by indentation. Procedure parameters are not shown here; detailed explanations are in the procedure listing.

```
Program Pest_Monitoring_System
    function read_response
    procedure response_unrecognized
    function file_exist

    {$I str_proc.pas }
        function delete_blanks
        function upper_case
        function last_characters_in_string
        function extract_alphabetic_char

    {$I siteproc.pas }
        function found_in_directory
        procedure search_for_site_in_directory
        procedure increment_to_next_site
        procedure fetch_site_name
        procedure valid_weather_site
        procedure fetch_master_directory
        function valid_trap_site
        procedure next_trap_site
        procedure fetch_pest_trapped_at_some_site

    {$I dateproc.pas }
        function Julian_to_numeric_date
        function date_string_from_numeric
        function date_string_from_Julian
        function calculate_Julian_date
    function numeric_date_to_Julian
    procedure CC_Get_DOS_Time
    procedure Julian_date_from_DOS
    procedure fetch_Julian_date
procedure read_BUGWATCH_parameters
procedure welcome
procedure display_command_menu
procedure sign_off

{$I inp-wthr.pas }
    procedure input_weather
        procedure fetch_weather_data
        procedure user_entered_weather_data

{$I inp-trap.pas }
    procedure input_trap_catch
        procedure next_trapped_pest
        procedure fetch_trap_data
        procedure user_entered_trap_data

{$I calculat.pas }
    procedure calculate_lifestage_development
        procedure fetch_weather_data_real
        function daily_development
            function piecewise_sine_curve
            function interpolate_development_rate
        procedure active_lifestage_development
        procedure read_MDL_file
    procedure calculate_trap_catch_statistics
        procedure calculate_summary
    procedure calculations

{$I printrep.pas }
    procedure print_report
        procedure get_yes_no_reply
        procedure fetch_wanted_reports
        procedure construct_list_of_all_sites
        procedure construct_list_of_requests
        procedure report_header
        procedure report_trailer
        procedure trap_catch_report
        procedure lifestage_development_report
        procedure population_events_report
        procedure print_list_of_reports

{$I regional.pas }
    procedure print_regional_reports
        procedure get_yes_no_reply
        procedure advance_to_next_region
        procedure construct_list_of_all_regions
        procedure construct_list_of_requests
```

```
            procedure make_line_prefixes
            procedure region_report_header
            procedure region_report_trailer
            procedure regional_development_report
            procedure print_list_of_regional_reports

    {$I graftrap.pas }
        procedure graph_trap_catch
            function do_yes_no_reply
            procedure get_yes_no_reply
            procedure fetch_wanted_graphs
            procedure construct_plot_list_of_all_sites
            procedure vertical_tick_mark
            procedure plot_single_graph
                procedure calculate_dot_position
                procedure get_trap_data
                procedure blank_graph
                    procedure write_month_abbreviation
                    procedure write_x_axis_tick_mark
                procedure trap_count_plot
                    procedure draw_heavy_line
            procedure construct_plot_list_of_requests
            procedure plot_list_of_graphs

{ End of program Pest_Monitoring_System }
```

2.2 CALCULATIONS

Procedure Calculate performs the calculations on all of the sites; the user has no choice. The calculations are made from the earlier of the earliest weather date changed and the beginning of the current lifestage.

Lifestage development rate

We use a sine curve to interpolate between successive daily temperature maxima and minima. This results in three separate sine curve segments to describe the daily temperature curve. The disadvantage is that this method requires data from three days. The method of Baskerville & Emin [1969] uses one day's data and could substitute for ours, but it is less accurate.

The function daily_development uses numerical integration by Simpson's rule to calculate developmental units. A higher order numerical method would improve accuracy, but this does not seem to be necessary.

Appendix 1 contains a complete description of the development rate function.

Trap catch

The output specifications require trap count data be printed starting with the most recent date. Storing the data in reverse date order facilitates the programming of the print report procedures.

Since trap records are stored in reverse date order a cumulative sum cannot be calculated at the same time as the daily trap sums. Thus, the trap count cumulative sum is calculated in its own calculation block at the end of procedure calculate_summary.

2.3 HARDCOPY OUTPUT

The print report module stores print requests until all are entered and then begins the actual printing. This way the user does not have to wait for the printout of one report before requesting another.

The graphs module plots line graphs of average trap count versus day on the screen for each pest at the user requested site. The program requests whether or not a hardcopy of the graph is desired. Requests for hardcopy graphs are treated in the same manner as in the print report module.

The file BUGSYS.PAR contains parameters which partly determine graph layout. For instance, the minimum and maximum on the x-axis are set by parameters in this file. However, the y-axis minimum is always zero and the maximum is computed by the program. The file description section of this manual (section 3) contains more information on these parameters.

Data points which lie outside the domain are not plotted. Missing points are represented by a double line.

2.4 USER INPUT

There is minimal error checking in the data entry modules. The temperature module does not check for unreasonable temperatures. Also, the procedure does not check that the minimum is less than the maximum temperature. This is deliberate. The minimum refers to the temperature at 6 am and the maximum refers to the temperature at 3 pm. Usually the morning temperature (min) is less than the afternoon temperature (max); however, the temperature may actually fall throughout the day so that the 6 am temperature (min) is in fact higher than the 3 pm temperature (max).

The trap catch module does not check for negative trap counts nor does it check that the count is unreasonably large. The word "missing" is the only valid nonnumeric input.

Any errors in input can be corrected by returning to the appropriate data entry point and re-entering the data.

3. FILES

All BUGWATCH files are stored in the subdirectory C:\OMAF-A-C. The subdirectory organization of the BUGWATCH system on the disk follows.

Several files are repeated in different subdirectories. For example, the subdirectory STUPFILE contains all the files of the subdirectory INITSITE and also contains an extra copy of MASTER.DIR.

The YEARnnnn subdirectory contains a master directory of all monitoring sites for that year and a subdirectory INITSITE which contains template data files. It also contains monitoring site subdirectories which contain the actual BUGWATCH data files. In other words, the YEARnnnn subdirectory groups BUGWATCH monitoring sites by year. This allows each year to be independent of other years with respect to monitoring sites, to which data are gathered, and to the developmental parameters of the pests. In addition, this organization facilitates file backup by season. Finally, we expect that year-by-year comparisons will be easier to implement using this type of file organization.

Site subdirectories contain files for storing weather data, pest trap counts, pest lifestage development and pest development rate parameters. The exact set of files depends on which data are gathered at the site. Weather and pest trap counts can vary from site to site and so each site must contain its own weather data, trap count data and pest development files. We repeated the pest lifestage development rate file in each weather monitoring site to allow for geographic differences in development rates (Trimble [1983]); at this time all development rate parameters are constant with respect to location.

3.1. DIRECTORY LISTING

```
Directory of  C:\OMAF-A-C

     INSTALLC BAT      1077
     BUGWATCH COM     56917
     BUGSYS   PAR       691
     UTILITY  COM     34780
     STARTDEL COM     18484
     STARTUP  BAT        46
     DSTARTNW BAT        18
     REMOVE   BAT        46
     DSTOPOLD BAT        17
     PROGFILE      <DIR>
     STUPFILE      <DIR>
```

```
MANUALS      <DIR>
YEAR1989     <DIR>

Directory of  C:\OMAF-A-C\PROGFILE

    BUGWATCH PAS    17190
    STR_PROC PAS     3094
    SITEPROC PAS    18918
    DATEPROC PAS    10183
    INP-WTHR PAS    14130
    INP-TRAP PAS    21332
    CALCULAT PAS    46057
    PRINTREP PAS    34697
    REGIONAL PAS    29202
    GRAFTRAP PAS    39797
    FILEPROC PAS     7264
    UTILPROC PAS    27495
    UTILITY  PAS    32317
    STARTDEL PAS    17292

Directory of  C:\OMAF-A-C\STUPFILE

    CODLING  TRP       58
    OFMOTH   TRP      100
    AMAGGOT  TRP       65
    OBLR     TRP       44
    RBLR     TRP       44
    STLM     TRP       44
    CODLING  MDL      166
    AMAGGOT  MDL      249
    AORNIGIS MDL      249
    ERM      MDL      166
    STLM     MDL      498
    CODLING  RAT      966
    AORNIGIS RAT     1344
    AMAGGOT  RAT     1344
    STLM     RAT     2478
    ERM      RAT      966
    WEATHER  DAT     6588
    WEATHER  FLG     2930
    WEATHER  INF        8
    MASTER   87      6819
    MASTER   DIR     ( variable size )

Directory of  C:\OMAF-A-C\manuals

    USER     LIS    39561
    PERCENT  LIS     8272
    MANAGER  LIS    29576
```

```
Directory of  C:\OMAF-A-C\YEAR1989

     MASTER   DIR      1006
     INITSITE      <DIR>
     SITE001       <DIR>
     SITE002       <DIR>
        .
        .
        .


     Directory of  C:\OMAF-A-C\YEAR1989\INITSITE

          CODLING  TRP        58
          OFMOTH   TRP       100
          AMAGGOT  TRP        65
          OBLR     TRP        44
          RBLR     TRP        44
          STLM     TRP        44
          CODLING  MDL       166
          AMAGGOT  MDL       249
          AORNIGIS MDL       249
          ERM      MDL       166
          STLM     MDL       498
          CODLING  RAT       966
          AORNIGIS RAT      1344
          AMAGGOT  RAT      1344
          STLM     RAT      2478
          ERM      RAT       966
          WEATHER  DAT      6588
          WEATHER  FLG      2930
          WEATHER  INF         8
          MASTER   87       6819

     Directory of  C:\OMAF-A-C\YEAR1989\SITE001
```

This subdirectory and all other site subdirectories contain the weather or pest file(s) according to the data gathered at the site. The templates of these files come from the subdirectory INITSITE. The file MASTER.87 does not appear in any site subdirectories.

3.2 DIRECTORY AND DATAFILE DESCRIPTION

Some files belong to the set of BUGWATCH Manager Utilities and will not be described in this manual. The program source code files are presented in the annexes. The files appear in the same order as they appear in the file listing above.

C:\OMAF-A-C <DIR>

The OMAF-A-C subdirectory contains the executable BUGWATCH program file, the parameter file BUGSYS.PAR and BUGWATCH system utilities.

C:\OMAF-A-C\INSTALLC.BAT

This batch procedure is created from the distribution floppy disk. Execution of this file automatically installs the BUGWATCH system onto hard disk drive C: and builds a YEARnnnn subdirectory tree for the current year (as stored in the system clock) from the MASTER.DIR included with the distribution files.

C:\OMAF-A-C\BUGWATCH.COM

The executable BUGWATCH program file.

C:\OMAF-A-C\BUGSYS.PAR

This file contains some formatting and setup parameters for the BUGWATCH program. The listing of BUGSYS.PAR follows.

```
13.0          ! width of paper in inches
12            ! characters per inch on printer
27 77 -1      ! printer initialization sequence
27 80 -1      ! printer termination sequence
7             ! HiRes ( high resolution ) graphics screen color
0401 0731     ! apple maggot                 start and end
0401 0831     ! apanteles ornigis            month-day for
0401 0630     ! codling moth                 trap count graphics
0401 0630     ! European red mite            on screen and printer
0401 0630     ! oblique banded leaf roller
0401 0630     ! oriental fruit moth
0401 0630     ! red banded leaf roller
0401 0930     ! tentiform leaf miner
```

The printer initialization and termination sequences are for an Epson FX-100 printer.

C:\OMAF-A-C\UTILITY.COM

The executable BUGWATCH system manager utility file.

C:\OMAF-A-C\STARTDEL.COM

The executable program which creates either of the STARTNEW or the STOPOLD batch files. STARTNEW builds the new season's YEARnnnn subdirectory, i.e. it STARTs a NEW year. STOPOLD removes last season's YEARnnnn subdirectory.

C:\OMAF-A-C\STARTUP.BAT

The batch procedure which invokes the STARTDEL program to create this season's YEARnnnn subdirectory. This is done once at the beginning of the new season.

C:\OMAF-A-C\DSTARTNW.BAT

The batch procedure to delete the STARTNEW.BAT file. This is done to free disk space.

C:\OMAF-A-C\REMOVE.BAT

The batch procedure which invokes the STARTDEL program to remove last season's YEARnnnn subdirectory. This is done at the end of the season in order to free disk space.

C:\OMAF-A-C\DSTOPOLD.BAT

The batch procedure to delete the STOPOLD.BAT file. This is done to free disk space.

C:\OMAF-A-C\PROGFILE <DIR>

The PROGFILE subdirectory contains the Pascal source code files for the BUGWATCH program and BUGWATCH system manager utility.

C:\OMAF-A-C\PROGFILE\BUGWATCH.PAS

This file contains the BUGWATCH main calling program.

C:\OMAF-A-C\PROGFILE\STR_PROC.PAS

STR_PROC.PAS contains string handling functions.

C:\OMAF-A-C\PROGFILE\SITEPROC.PAS

SITEPROC.PAS contains procedures and functions which read the MASTER.DIR external disk file and the internal program directory array.

C:\OMAF-A-C\PROGFILE\DATEPROC.PAS

DATEPROC.PAS contains date handling procedures and functions. The procedures fetch the time stamp from DOS. The functions convert dates from one format into another, e.g. Julian date to month-day.

C:\OMAF-A-C\PROGFILE\INP-WTHR.PAS

INP-WTHR.PAS contains the input_weather procedure which handles all weather data input.

C:\OMAF-A-C\PROGFILE\INP-TRAP.PAS

INP-TRAP.PAS contains the input_trap_catch procedure which handles all trap catch data input.

C:\OMAF-A-C\PROGFILE\CALCULAT.PAS

CALCULAT.PAS contains the procedures which calculate pest lifestage development from weather data and which calculate trap catch statistics from the trap catch data.

C:\OMAF-A-C\PROGFILE\PRINTREP.PAS

PRINTREP.PAS contains the print_report procedure which handles requests for textual reports (trap catch, lifestage development and lifestage events) and which handles the printing of the reports.

C:\OMAF-A-C\PROGFILE\REGIONAL.PAS

REGIONAL.PAS contains the print_regional_reports procedure which handles requests for regional reports and which handles the printing of regional reports. A regional report contains the pest lifestage development information on all sites within a region.

C:\OMAF-A-C\PROGFILE\GRAFTRAP.PAS

GRAFTRAP.PAS contains the graph_trap_catch procedure which handles requests for graphs of pest trap catches and which handles the printing of the graphs.

C:\OMAF-A-C\PROGFILE\FILEPROC.PAS

FILEPROC.PAS contains file and directory handling procedures and functions.

C:\OMAF-A-C\PROGFILE\UTILPROC.PAS

UTILPROC.PAS contains the utility procedures which perform the actual utility tasks, e.g. adding a new site to the current season's set of BUGWATCH monitoring sites.

C:\OMAF-A-C\PROGFILE\UTILITY.PAS

This file contains the BUGWATCH System Manager's main calling program.

C:\OMAF-A-C\PROGFILE\STARTDEL.PAS

This file contains the main calling program for STARTing a new season or DELeting the previous one. STARTing a new season consists of building a new YEARnnnn subdirectory for the season. The new season's YEARnnnn subdirectory is based on the previous season. In particular, the previous season's MASTER.DIR file and INITSITE subdirectory are used for the new season. DELeting a season consists of deleting the YEARnnnn subdirectory.

C:\OMAF-A-C\STUPFILE <DIR>

STUPFILE stands for SeTUPFILE. This subdirectory resides on the distribution floppy disk and contains all the data files needed to setup a functioning BUGWATCH YEARnnnn subdirectory. The STUPFILE files are used to create a new INITSITE sub-subdirectory. This is required if BUGWATCH is being installed on a new machine or if the INITSITE files are damaged. These files are also used by the STARTDEL procedure to build a YEARnnnn subdirectory if the previous season was deleted before the new one was built.

C:\OMAF-A-C\STUPFILE\*.TRP

A *.TRP file stands for TRaP catch file. There is one record for each day on which trap counts are taken. The general format of a record in a *.TRP file is as follows...

```
MMDDaaaaa.aCCCCCCNNNGGGGGGGGttttttt ( more "ttttttt" )
```

MM is an integer representation of the month.

DD is an integer representation of the day.

aaaaa.a is the average trap count for the day rounded to the nearest tenth.

CCCCCC is the total trap count for the day.
NNN is the number of traps from which counts were taken on this day.

GGGGGGGG is the grand total trap count up to this day.

ttttttt is the trap count for an individual trap. The first "ttttttt" field corresponds to trap 1; the second "ttttttt" field corresponds to trap 2; more "ttttttt" fields may follow.

The first record in the file is a dummy record consisting of all zeros. The number of "ttttttt" fields in this record equals the maximum number of traps for the pest at the site. For example, a pest for which 4 traps are used has the following dummy record ...

```
 000     0.0     0  0       0      0      0      0      0
```

The BUGWATCH program determines the maximum number of traps used for a pest at a site by counting the number of trap fields in the dummy record.

C:\OMAF-A-C\STUPFILE\CODLING.TRP

This file contains the codling moth trap data. The maximum number of traps is 4.

C:\OMAF-A-C\STUPFILE\OFMOTH.TRP

This file contains the oriental fruit moth trap data. The maximum number of traps is 10.

C:\OMAF-A-C\STUPFILE\AMAGGOT.TRP

This file contains the apple maggot trap data. The maximum number of traps is 5.

C:\OMAF-A-C\STUPFILE\OBLR.TRP

This file contains the oblique banded leaf roller trap data. The maximum number of traps is 2.

C:\OMAF-A-C\STUPFILE\RBLR.TRP

This file contains the red banded leaf roller trap data. The maximum number of traps is 2.

C:\OMAF-A-C\STUPFILE\STLM.TRP

This file contains the spotted tentiform leaf miner trap data. The maximum number of traps is 2.

C:\OMAF-A-C\STUPFILE\*.MDL

A *.MDL file stands for MoDeL. The *.MDL file contains the lifestage development predictions based on the development models. There is one record for each lifestage in the model. A record in a *.MDL file consists of 13 fields.

Field 1 is a 1 column character field. The character code indicates the state of the lifestage; H for Hibernating, A for Active or currently developing, P for Past. At the beginning of the season all lifestages are Hibernating; at the end of the season all lifestages are Past. The lifestage is Active during the time developmental developmental units are accumulating.

Field 2 is a 2 column integer field. The integer represents the lifestage generation.

Field 3 is a 30 column character field storing the name of the lifestage.

Fields 4, 6, and 8 to 13 are 4 column integer fields containing Julian dates.

Fields 5 and 7 are 8 column real number fields; the real numbers use 4 places to the right of the decimal place.

Field 4 contains the most recent Julian date to which actual weather data were used to calculate pest development. Field 5 contains the fraction of lifestage development to that date. Note that field 4 may contain a Julian date earlier than the most recent date on which actual weather data have been entered. This will be true until the user issues a Calculate command. After the Calculate is done the Julian date in field 4 will match the most recent date on which actual weather data have been entered.

The real numbers in fields 5 and 7 represent the fraction of lifestage development. 0.0000 means no developmental units required for development have accumulated (no development has occurred). 1.0000 means all the developmental units required for complete development have been accumulated.

Fields 6 and 7 contain the date and fraction of lifestage development for predicted weather data.

Field 8 contains the Julian date on which the model equations predict the lifestage begins development. Field 9 contains the Julian date on which the field observations indicate the lifestage begins development (fraction = 0.0000).

Fields 10 and 11 contain the Julian dates on which the model and observations indicate the lifestage completes development (fraction = 1.0000).

Fields 12 and 13 contain the Julian dates on which the model and observations indicate the lifestage has past full development by 10 percent (fraction = 1.1000).

The current version of BUGWATCH does not use fields 9, 11, and 13 (field observations). Designing the *.MDL file with these fields will facilitate integration of field observations with model predictions in future versions of BUGWATCH.

C:\OMAF-A-C\STUPFILE\CODLING.MDL

This file contains the codling moth lifestage development data. The number of lifestages is 2.

C:\OMAF-A-C\STUPFILE\AMAGGOT.MDL

This file contains the apple maggot lifestage development data. The number of lifestages is 3.

C:\OMAF-A-C\STUPFILE\AORNIGIS.MDL

This file contains the Apanteles ornigis lifestage development data. The number of lifestages is 3.

C:\OMAF-A-C\STUPFILE\ERM.MDL

This file contains the European red mite lifestage development data. The number of lifestages is 2.

C:\OMAF-A-C\STUPFILE\STLM.MDL

This file contains the spotted tentiform leaf miner lifestage development data. The number of lifestages is 6.

C:\OMAF-A-C\STUPFILE\*.RAT

A *.RAT file stands for RATe. The *.RAT file contains the lifestage development rate functions in a matrix format. Matrix element (T, L) equals the development rate of lifestage L at temperature T. Temperatures range from -1 to 40 degrees Celsius. There is one record for each temperature, and one column for each lifestage. The general format of a record in a *.RAT file is as follows...

TTTfff.fffff (may have more "fff.fffff" fields)

TTT is the temperature. Temperatures are integer values.

fff.fffff is the development rate at the temperature TTT. The first "fff.fffff" field corresponds to the first lifestage; the second "fff.fffff" field corresponds to lifestage 2; more fields (lifestages) may follow.

The first record in this file with a temperature value of -1 is a dummy record. The number of "fff.fffff" fields in this record equals the number of lifestages modeled at the site. For example, a pest with two modeled lifestages has the following dummy record ...

```
 -1  0.00000  0.00000
```

C:\OMAF-A-C\STUPFILE\CODLING.RAT

This file contains the codling moth lifestage development rate data. The number of lifestages is 2.

C:\OMAF-A-C\STUPFILE\AORNIGIS.RAT

This file contains the Apanteles ornigis development rate data. The number of lifestages is 3.

C:\OMAF-A-C\STUPFILE\AMAGGOT.RAT

This file contains the apple maggot development rate data. The number of lifestages is 3.

C:\OMAF-A-C\STUPFILE\STLM.RAT

This file contains the spotted tentiform leaf miner development rate data. The number of lifestages is 6.

C:\OMAF-A-C\STUPFILE\ERM.RAT

This file contains the European red mite development rate data. The number of lifestages is 2.

C:\OMAF-A-C\STUPFILE\WEATHER.DAT

This file contains the maximum and minimum temperatures for each day of the year. There is one record for each day. The format of each record is ...

MMDDxxxx.xyyyy.y

MM is the integer representation of the month. DD is the integer representation of the day. xxxx.x is the maximum temperature rounded to the closest tenth of a degree Celsius. yyyy.y is the minimum temperature rounded to the closest tenth of a degree Celsius.

C:\OMAF-A-C\STUPFILE\WEATHER.FLG

This file contains the type flags for the weather data corresponding to each day of the year. There is one record for each day. The format of each record is ...

MMDD F

MM is the integer representation of the month. DD is the integer representation of the day. F is a letter code indicating the data type; N for Normal, H for Historical, A for Actual, and P for Predicted.

C:\OMAF-A-C\STUPFILE\WEATHER.INF

This file contains just one record which contains the earliest Julian date for which weather data were modified subsequent to the last Calculate task. Initially, this date is 366.

C:\OMAF-A-C\STUPFILE\MASTER.87

This file is the MASTER.DIR file for the 1987 season.

C:\OMAF-A-C\STUPFILE\MASTER.DIR

MASTER.DIR contains data about each site monitored. These data completely specify a site. There is one record for each site. Each record consists of 13 character fields. The end-of-file marker for MASTER.DIR occurs at the end of the last line.

The fields have the following meaning.

field	width	contents
1	30	full name of the site
2	9	8 character abbreviation of the site name
3	16	name of the region in which the site resides
4	9	name of nearest weather site

The code in fields 5 to 13 indicates whether or not the site traps (gathers) or models data.

code	meaning
T	TRAPPED only
M	MODELED only
B	both TRAPPED and MODELED
other	neither TRAPPED nor MODELED

Notes:

1. For weather, "TRAPPED" means weather data are gathered. In this version of BUGWATCH only maximum and minimum air temperatures are gathered. "MODELED" means weather data are modeled. This version of BUGWATCH does not model weather data.

2. For the insects, "TRAPPED" means trap counts are gathered, and "MODELED" means lifestage development is modeled.

field	width	weather or pest
5	2	weather
6	2	apple maggot
7	2	Apanteles ornigis
8	2	codling moth
9	2	European red mite
10	2	oblique banded leaf roller
11	2	oriental fruit moth
12	2	red banded leaf roller
13	2	tentiform leaf miner

MASTER.DIR is read into an internal array at program startup. The program uses this internal copy of MASTER.DIR to obtain site specific information.

The monitoring fields (5 to 13) are used for two reasons. The first is to reduce disk accesses. BUGWATCH must determine which lifestage development or trap count calculations must be performed or if weather data are gathered. This could be done by searching the disk for the appropriate *.TRP or *.MDL or WEATHER files. Consulting the internal MASTER.DIR array for this information is faster. The second reason is to allow the BUGWATCH system manager to disable monitoring in mid-season without deleting any data files; instead, the monitoring flag is set to OFF. A monitoring flag set to OFF tells BUGWATCH to ignore the pest or weather. This is just as effective as deleting the data files, but has the advantage of preserving any data gathered. This is even more advantageous if the system manager wants to reactivate monitoring later in the season.

C:\OMAF-A-C\MANUALS\USER.LIS

This file contains the User's Guide. The Guide instructs the user on how to operate the BUGWATCH program; examples are included.

C:\OMAF-A-C\MANUALS\PERCENT.LIS

This file contains a detailed explanation describing the interpretation of the percent pest lifestage development which is printed in the BUGWATCH lifestage development report.

C:\OMAF-A-C\MANUALS\MANAGER.LIS

This file contains the BUGWATCH System Manager's Guide. The Guide explains how to modify the MASTER.DIR and pest developmental rate files.

C:\OMAF-A-C\YEAR1989\MASTER.DIR

The format of this file is the same as that of C:\OMAF-A-C\STUPFILE\MASTER.DIR. The record contents may differ depending on the actual sites used and the data gathered or modeled for the 1989 season.

C:\OMAF-A-C\YEAR1989\INITSITE <DIR>

This subdirectory contains the template files for the weather data, pest trap catch data, and predicted lifestage development data. It also contains the files which store the lifestage development rate parameters.

The template files for weather data have 366 records, one for each day of a leap year.

The trap catch data template files for a specific pest contain one record which gives BUGWATCH the format for trap catch data for that pest. There is a separate template file for each pest for which trap counts are monitored. Unlike weather data template files, trap catch template files do not contain a record for each day of the year. There are two reasons for this. The first is the unnecessary amount of disk space that would be used; trap data are not gathered every day. The second is that BUGWATCH expects trap catch data to be stored consecutively with no blank records. Since trap catches are gathered on variable days of the week the template file cannot assume a set of standard data gathering dates.

C:\OMAF-A-C\YEAR1989\SITE001 <DIR>

This subdirectory and all other site subdirectories contain the weather or pest file(s) according to the data gathered at the site. The templates of these files are copied from the subdirectory INITSITE at the start of the season. The file MASTER.87 does not appear in any site subdirectories.

Site subdirectories are usually named with the 8 character abbreviation of the site name rather than numbered Site 001, Site 002, etc.

The data fields in the weather data files are updated with actual data using the Input Weather task of BUGWATCH.

Each new set of trap data adds a record to the trap catch file.

4. PROCEDURE - DATAFILE MATRIX

This section shows the access relationships among the procedures and datafiles.

The matrix entries are one of R, W, or - (hyphen). These entries mean Read access, Write access, and No access respectively.

Note :

(1) The function file_exist reads file headers from the disk directory to determine whether or not the file specified in the input parameter exists. File_exist does not access files in the sense of reading or writing data.

Procedure or function	Data file							
	MASTER DIR	BUGSYS PAR	* TRP	* MDL	* RAT	WEATHER DAT	WEATHER FLG	WEATHER INF
Pest_Monitoring_System	-	-	-	-	-	-	-	-
read_response	-	-	-	-	-	-	-	-
response_unrecognized	-	-	-	-	-	-	-	-
file_exist	-	-	-	-	(1)	-	-	-
{$I str_proc.pas }								
delete_blanks	-	-	-	-	-	-	-	-
upper_case	-	-	-	-	-	-	-	-
last_characters_in_string	-	-	-	-	-	-	-	-
extract_alphabetic_char	-	-	-	-	-	-	-	-
{$I siteproc.pas }								
found_in_directory	-	-	-	-	-	-	-	-
search_for_site_in_directory	-	-	-	-	-	-	-	-
increment_to_next_site	-	-	-	-	-	-	-	-
fetch_site_name	-	-	-	-	-	-	-	-
valid_weather_site	-	-	-	-	-	-	-	-
fetch_master_directory	R	-	-	-	-	-	-	-
valid_trap_site	-	-	-	-	-	-	-	-
next_trap_site	-	-	-	-	-	-	-	-
fetch_pest_trapped_at_some_site	-	-	-	-	-	-	-	-
{$I dateproc.pas }								
Julian_to_numeric_date	-	-	-	-	-	-	-	-
date_string_from_numeric	-	-	-	-	-	-	-	-
date_string_from_Julian	-	-	-	-	-	-	-	-
calculate_Julian_date	-	-	-	-	-	-	-	-
numeric_date_to_Julian	-	-	-	-	-	-	-	-
CC_Get_DOS_Time	-	-	-	-	-	-	-	-

Procedure or function	Data file							
	MASTER DIR	BUGSYS PAR	* TRP	* MDL	* RAT	WEATHER DAT	WEATHER FLG	WEATHER INF
Julian_date_from_DOS	-	-	-	-	-	-	-	-
fetch_Julian_date	-	-	-	-	-	-	-	-
read_BUGWATCH_parameters	-	R	-	-	-	-	-	-
welcome	-	-	-	-	-	-	-	-
display_command_menu	-	-	-	-	-	-	-	-
sign_off	-	-	-	-	-	-	-	-
{$I inp-wthr.pas }								
input_weather	-	-	-	-	-	-	-	-
fetch_weather_data	-	-	-	-	-	R	R	-
user_entered_weather_data	-	-	-	-	-	W	W	RW
{$I inp-trap.pas }								
input_trap_catch	-	-	-	-	-	-	-	-
next_trapped_pest	-	-	-	-	-	-	-	-
fetch_trap_data	-	-	R	-	-	-	-	-
user_entered_trap_data	-	-	W	-	-	-	-	-
{$I calculat.pas }								
calculate_lifestage_development	-	-	-	W	R	-	-	RW
fetch_weather_data_real	-	-	-	-	-	R	R	-
daily_development	-	-	-	-	-	-	-	-
piecewise_sine_curve	-	-	-	-	-	-	-	-
interpolate_development_rate	-	-	-	-	-	-	-	-
active_lifestage_development	-	-	-	-	-	-	-	-
read_MDL_file	-	-	-	R	-	-	-	-
calculate_trap_catch_statistics	-	-	-	-	-	-	-	-
calculate_summary	-	-	RW	-	-	-	-	-
calculations	-	-	-	-	-	-	-	-
{$I printrep.pas }								
print_report	-	-	-	-	-	-	-	-
get_yes_no_reply	-	-	-	-	-	-	-	-
fetch_wanted_reports	-	-	-	-	-	-	-	-
construct_list_of_all_sites	-	-	-	-	-	-	-	-
construct_list_of_requests	-	-	-	-	-	-	-	-
report_header	-	-	-	-	-	-	-	-
report_trailer	-	-	-	-	-	-	-	-
trap_catch_report	-	-	R	-	-	-	-	-
lifestage_development_report	-	-	-	R	-	-	-	-

Procedure or function	Data file							
	MASTER DIR	BUGSYS PAR	* TRP	* MDL	* RAT	WEATHER DAT	WEATHER FLG	WEATHER INF
population_events_report	-	-	-	R	-	-	-	-
print_list_of_reports	-	-	-	-	-	-	-	-
{$I regional.pas }								
print_regional_reports	-	-	-	-	-	-	-	-
get_yes_no_reply	-	-	-	-	-	-	-	-
advance_to_next_region	-	-	-	-	-	-	-	-
construct_list_of_all_regions	-	-	-	-	-	-	-	-
construct_list_of_requests	-	-	-	-	-	-	-	-
make_line_prefixes	-	-	-	R	R	-	-	-
region_report_header	-	-	-	-	-	-	-	-
region_report_trailer	-	-	-	-	-	-	-	-
regional_development_report	-	-	-	R	-	-	-	-
print_list_of_regional_reports	-	-	-	-	-	-	-	-
{$I graftrap.pas }								
graph_trap_catch	-	-	-	-	-	-	-	-
do_yes_no_reply	-	-	-	-	-	-	-	-
get_yes_no_reply	-	-	-	-	-	-	-	-
fetch_wanted_graphs	-	-	-	-	-	-	-	-
construct_plot_list_of_all_sites	-	-	-	-	-	-	-	-
vertical_tick_mark	-	-	-	-	-	-	-	-
plot_single_graph	-	-	-	-	-	-	-	-
calculate_dot_position	-	-	-	-	-	-	-	-
get_trap_data	-	-	R	-	-	-	-	-
blank_graph	-	-	-	-	-	-	-	-
write_month_abbreviation	-	-	-	-	-	-	-	-
write_x_axis_tick_mark	-	-	-	-	-	-	-	-
trap_count_plot	-	-	-	-	-	-	-	-
draw_heavy_line	-	-	-	-	-	-	-	-
construct_plot_list_of_requests	-	-	-	-	-	-	-	-
plot_list_of_graphs	-	-	-	-	-	-	-	-

End of Matrix.

REFERENCES

Baskerville, G.L. and P. Emin, 1969.
"Rapid estimation of heat unit accumulation from maximum and minimum temperatures."
Ecology 50(3):514-517

Borland International Inc., 1985.
Turbo Pascal version 3.0 Reference Manual
4585 Scotts Valley Drive
Scotts Valley, CA, USA 95066

Hagley, E.A.C., R. Trottier, D.H.C. Herne, A. Hikichi, and A. Maitland, 1980.
"Pest Management in Ontario Apple Orchards"
Research Branch, Canada Department of Agriculture bulletin

Harcourt, D.G. and J.M. Yee, 1982.
"Polynomial Algorithm for Predicting the Duration of Insect Life Stages"
Environmental Entomology 11(3):581-584

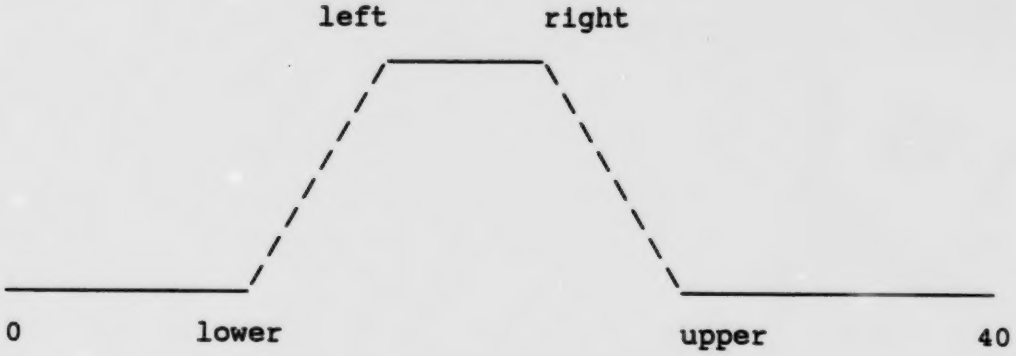
Koziol, A., 1978.
"Pest Management System"
Unpublished documentation.

Trimble, R.M., 1983.
"Diapause determination and the thermal requirements for postdiapause development in six Ontario populations of the spotted tentiform leafminer, Phyllonorycter blancardella (Lepidoptera : Gracillaridae)"
Canadian Entomologist 115:387-392

Trottier, R., 1977.
"An early pest warning system based on weather records"
Canadian Fruitgrower 33(5):11

APPENDIX 1. DEVELOPMENT RATE FUNCTION

The development rate function is the derivative with respect to temperature of the development function. The rate development functions are defined between 0 and 40 degrees Celsius. The general shape of the function for any lifestage of all pests follows.



The development rate is 0.0 on the interval [0, lower). On [lower, left) the development rate increases linearly with respect to temperature. The rate rises to its maximum at temperature equals left and remains constant until temperature equals right. On [right, upper) the development rate decreases linearly with respect to temperature. The rate falls to 0.0 at temperature equals upper and remains at 0.0 beyond this value.

The utility modrat allows the manager to change the values for lower, left, right, and upper. The general shape of the function cannot be changed. These values must satisfy

$$0 <= lower < left <= right < upper <= 40$$

for the program to function properly. There is no error checking in the utility for this inequality. The slope of the line between lower and left is determined by the number of developmental units needed for complete development of the lifestage (Harcourt and Yee [1982]). This value corresponds approximately to the ACTS (Agriculture Canada Time Sharing system scientific software) degree days and is input by the BUGWATCH manager.

APPENDIX 2. LISTING OF DATAFILES IN SUBDIRECTORY STUPFILE

```
CODLING.TRP
-----------
 000    0.0     0  0      0     0     0     0     0

OFMOTH.TRP
----------
 000    0.0     0  0      0     0     0     0     0     0     0     0     0     0     0

AMAGGOT.TRP
-----------
 000    0.0     0  0      0     0     0     0     0     0

OBLR.TRP
--------
 000    0.0     0  0      0     0     0

RBLR.TRP
--------
 000    0.0     0  0      0     0     0

STLM.TRP
--------
 000    0.0     0  0      0     0     0

CODLING.MDL
-----------
H 1first catch                        1  0.0000   1  0.0000  62 366 366 366 366 366
H 1first hatch                        1  0.0000   1  0.0000 366 366 366 366 366 366

AMAGGOT.MDL
-----------
H 1first immature female (Y)          1  0.0000   1  0.0000  62 366 366 366 366 366
H 1first mature female (R)            1  0.0000   1  0.0000 366 366 366 366 366 366
H 1peak catch                         1  0.0000   1  0.0000 366 366 366 366 366 366

AORNIGIS.MDL
------------
H 0first emergence (overwinter)       1  0.0000   1  0.0000  62 366 366 366 366 366
H 1first emergence (summer 1)         1  0.0000   1  0.0000 366 366 366 366 366 366
H 2first emergence (summer 2)         1  0.0000   1  0.0000 366 366 366 366 366 366

ERM.MDL
-------
H 1first hatch                        1  0.0000   1  0.0000  62 366 366 366 366 366
H 150% hatch                          1  0.0000   1  0.0000 366 366 366 366 366 366
```

```
STLM.MDL
--------
H 0first emergence (overwinter)     1  0.0000   1  0.0000  62 366 366 366 366 366
H 0first instar larva (overwin.)    1  0.0000   1  0.0000 366 366 366 366 366 366
H 1first emergence (summer 1)       1  0.0000   1  0.0000 366 366 366 366 366 366
H 1first instar larva (summer 1)    1  0.0000   1  0.0000 366 366 366 366 366 366
H 2first emergence (summer 2)       1  0.0000   1  0.0000 366 366 366 366 366 366
H 2first instar larva (summer 2)    1  0.0000   1  0.0000 366 366 366 366 366 366
```

```
CODLING.RAT
-----------

 -1  0.00000  0.00000
  0  0.00000  0.00000
  1  0.00000  0.00000
  2  0.00000  0.00000
  3  0.00000  0.00000
  4  0.00000  0.00000
  5  0.00000  0.00000
  6  0.00000  0.00000
  7  0.00000  0.00000
  8  0.00000  0.00000
  9  0.00000  0.00000
 10  0.00000  0.00000
 11  0.00000  0.00000
 12  0.01000  0.01000
 13  0.02000  0.02000
 14  0.03000  0.03000
 15  0.04000  0.04000
 16  0.05000  0.05000
 17  0.06000  0.06000
 18  0.07000  0.07000
 19  0.08000  0.08000
 20  0.09000  0.09000
 21  0.10000  0.10000
 22  0.11000  0.11000
 23  0.12000  0.12000
 24  0.13000  0.13000
 25  0.14000  0.14000
 26  0.15000  0.15000
 27  0.16000  0.16000
 28  0.17000  0.17000
 29  0.17000  0.17000
 30  0.17000  0.17000
 31  0.17000  0.17000
 32  0.17000  0.17000
 33  0.11333  0.11333
 34  0.05667  0.05667
 35  0.00000  0.00000
 36  0.00000  0.00000
 37  0.00000  0.00000
 38  0.00000  0.00000
 39  0.00000  0.00000
 40  0.00000  0.00000
```

```
AORNIGIS.RAT
------------

 -1  0.00000  0.00000  0.00000
  0  0.00000  0.00000  0.00000
  1  0.00000  0.00000  0.00000
  2  0.00000  0.00000  0.00000
  3  0.00000  0.00000  0.00000
  4  0.00000  0.00000  0.00000
  5  0.00000  0.00000  0.00000
  6  0.00000  0.00000  0.00000
  7  0.00000  0.00000  0.00000
  8  0.00000  0.00000  0.00000
  9  0.00000  0.00000  0.00000
 10  0.00000  0.00000  0.00000
 11  0.00600  0.00203  0.00150
 12  0.01600  0.00542  0.00400
 13  0.02600  0.00881  0.00650
 14  0.03600  0.01220  0.00900
 15  0.04600  0.01559  0.01150
 16  0.05600  0.01898  0.01400
 17  0.06600  0.02237  0.01650
 18  0.07600  0.02576  0.01900
 19  0.08600  0.02915  0.02150
 20  0.09600  0.03254  0.02400
 21  0.10600  0.03593  0.02650
 22  0.11600  0.03932  0.02900
 23  0.12600  0.04271  0.03150
 24  0.13600  0.04610  0.03400
 25  0.14600  0.04949  0.03650
 26  0.15600  0.05288  0.03900
 27  0.16600  0.05627  0.04150
 28  0.17600  0.05966  0.04400
 29  0.17600  0.05966  0.04400
 30  0.17600  0.05966  0.04400
 31  0.17600  0.05966  0.04400
 32  0.17600  0.05966  0.04400
 33  0.11733  0.03977  0.02933
 34  0.05867  0.01989  0.01467
 35  0.00000  0.00000  0.00000
 36  0.00000  0.00000  0.00000
 37  0.00000  0.00000  0.00000
 38  0.00000  0.00000  0.00000
 39  0.00000  0.00000  0.00000
 40  0.00000  0.00000  0.00000
```

AMAGGOT.RAT

```
 -1   0.00000   0.00000   0.00000
  0   0.00000   0.00000   0.00000
  1   0.00000   0.00000   0.00000
  2   0.00000   0.00000   0.00000
  3   0.00000   0.00000   0.00000
  4   0.00000   0.00000   0.00000
  5   0.00000   0.00000   0.00000
  6   0.00000   0.00000   0.00000
  7   0.00000   0.00000   0.00000
  8   0.00000   0.00000   0.00000
  9   0.00000   0.00000   0.00000
 10   0.00244   0.01000   0.00364
 11   0.00488   0.02000   0.00727
 12   0.00732   0.03000   0.01091
 13   0.00976   0.04000   0.01455
 14   0.01220   0.05000   0.01818
 15   0.01463   0.06000   0.02182
 16   0.01707   0.07000   0.02545
 17   0.01951   0.08000   0.02909
 18   0.02195   0.09000   0.03273
 19   0.02439   0.10000   0.03636
 20   0.02683   0.11000   0.04000
 21   0.02927   0.12000   0.04364
 22   0.03171   0.13000   0.04727
 23   0.03415   0.14000   0.05091
 24   0.03659   0.15000   0.05455
 25   0.03902   0.16000   0.05818
 26   0.04146   0.17000   0.06182
 27   0.04390   0.18000   0.06545
 28   0.04634   0.19000   0.06909
 29   0.04634   0.19000   0.06909
 30   0.04634   0.19000   0.06909
 31   0.04634   0.19000   0.06909
 32   0.04634   0.19000   0.06909
 33   0.03089   0.12667   0.04606
 34   0.01545   0.06333   0.02303
 35   0.00000   0.00000   0.00000
 36   0.00000   0.00000   0.00000
 37   0.00000   0.00000   0.00000
 38   0.00000   0.00000   0.00000
 39   0.00000   0.00000   0.00000
 40   0.00000   0.00000   0.00000
```

STLM.RAT

```
 -1  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  0  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  1  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  2  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  3  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  4  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  5  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  6  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  7  0.00750  0.00273  0.00082  0.00171  0.00076  0.00286
  8  0.03250  0.01182  0.00356  0.00743  0.00329  0.01238
  9  0.05750  0.02091  0.00630  0.01314  0.00582  0.02190
 10  0.08250  0.03000  0.00904  0.01886  0.00835  0.03143
 11  0.10750  0.03909  0.01178  0.02457  0.01089  0.04095
 12  0.13250  0.04818  0.01452  0.03029  0.01342  0.05048
 13  0.15750  0.05727  0.01726  0.03600  0.01595  0.06000
 14  0.18250  0.06636  0.02000  0.04171  0.01848  0.06952
 15  0.20750  0.07545  0.02274  0.04743  0.02101  0.07905
 16  0.23250  0.08455  0.02548  0.05314  0.02354  0.08857
 17  0.25750  0.09364  0.02822  0.05886  0.02608  0.09810
 18  0.28250  0.10273  0.03096  0.06457  0.02861  0.10762
 19  0.30750  0.11182  0.03370  0.07029  0.03114  0.11714
 20  0.33250  0.12091  0.03644  0.07600  0.03367  0.12667
 21  0.35750  0.13000  0.03918  0.08171  0.03620  0.13619
 22  0.38250  0.13909  0.04192  0.08743  0.03873  0.14571
 23  0.40750  0.14818  0.04466  0.09314  0.04127  0.15524
 24  0.43250  0.15727  0.04740  0.09886  0.04380  0.16476
 25  0.45750  0.16636  0.05014  0.10457  0.04633  0.17429
 26  0.48250  0.17545  0.05288  0.11029  0.04886  0.18381
 27  0.50750  0.18455  0.05562  0.11600  0.05139  0.19333
 28  0.53250  0.19364  0.05836  0.12171  0.05392  0.20286
 29  0.53250  0.19364  0.05836  0.12171  0.05392  0.20286
 30  0.53250  0.19364  0.05836  0.12171  0.05392  0.20286
 31  0.53250  0.19364  0.05836  0.12171  0.05392  0.20286
 32  0.53250  0.19364  0.05836  0.12171  0.05392  0.20286
 33  0.35500  0.12909  0.03890  0.08114  0.03595  0.13524
 34  0.17750  0.06455  0.01945  0.04057  0.01797  0.06762
 35  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
 36  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
 37  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
 38  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
 39  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
 40  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
```

```
ERM.RAT
-------

 -1   0.00000   0.00000
  0   0.00000   0.00000
  1   0.00000   0.00000
  2   0.00000   0.00000
  3   0.00000   0.00000
  4   0.00000   0.00000
  5   0.00000   0.00000
  6   0.00308   0.00421
  7   0.01077   0.01474
  8   0.01846   0.02526
  9   0.02615   0.03579
 10   0.03385   0.04632
 11   0.04154   0.05684
 12   0.04923   0.06737
 13   0.05692   0.07789
 14   0.06462   0.08842
 15   0.07231   0.09895
 16   0.08000   0.10947
 17   0.08769   0.12000
 18   0.09538   0.13053
 19   0.10308   0.14105
 20   0.11077   0.15158
 21   0.11846   0.16211
 22   0.12615   0.17263
 23   0.13385   0.18316
 24   0.14154   0.19368
 25   0.14923   0.20421
 26   0.15692   0.21474
 27   0.16462   0.22526
 28   0.17231   0.23579
 29   0.17231   0.23579
 30   0.17231   0.23579
 31   0.17231   0.23579
 32   0.17231   0.23579
 33   0.11487   0.15719
 34   0.05744   0.07860
 35   0.00000   0.00000
 36   0.00000   0.00000
 37   0.00000   0.00000
 38   0.00000   0.00000
 39   0.00000   0.00000
 40   0.00000   0.00000
```

```
WEATHER.DAT
-----------
 101    6.5   0.0
 102    2.0  -3.0
 103   -7.0  -9.0
 104    3.0 -13.0
 105   -1.5 -10.5
 106   -2.0 -14.0
 107    3.0  -6.0
 108    5.5  -2.0
 109    2.5  -4.5
 110    8.0  -2.0
 111    9.0   3.0
 112    1.5   0.0
 113    2.5  -3.0
 114    0.5  -1.0
 115    4.0  -2.5
 116    6.0  -8.0
 117   -3.5 -10.0
 118    5.0 -11.0
 119    6.5 -10.0
 120    0.5  -5.0
 121   -2.0  -7.5
 122    5.5 -10.0
 123    6.0  -4.5
 124    5.0  -0.5
 125    0.0  -6.0
 126   10.0  -8.0
 127   14.5   3.5
 128    5.0   2.5
 129    7.0   1.0
 130    2.0  -2.5
 131    3.5  -2.5
 201    1.5  -1.0
 202    0.5  -3.0
 203    3.0  -6.0
 204    6.5  -1.5
 205   11.0  -1.0
 206    6.0  -1.5
 207    6.0  -2.5
 208   -0.5  -4.0
 209   -2.0  -6.5
 210    6.0  -7.0
 211    2.0  -1.5
 212    5.0  -0.7
 213    8.5   0.0
 214   21.5   1.5
 215   19.0   8.0
 216   13.5   4.5
 217    3.0  -4.0
 218   22.0   0.0
```

219	14.0	2.0
220	23.0	4.5
221	28.0	8.5
222	26.0	13.0
223	28.5	14.0
224	22.5	11.5
225	16.5	9.0
226	15.5	6.0
227	4.5	0.0
228	10.0	-0.5
229	18.5	-0.5
301	-3.0	-13.0
302	1.0	-7.0
303	5.0	-4.0
304	2.0	-2.0
305	3.0	-3.0
306	0.0	-11.0
307	-10.0	-16.0
308	-5.0	-19.0
309	4.0	-5.0
310	14.0	-1.0
311	5.0	-4.0
312	2.0	-5.0
313	3.0	0.0
314	4.0	1.0
315	5.0	0.0
316	3.0	-1.0
317	2.0	-3.0
318	6.0	-2.0
319	14.0	-5.0
320	-5.0	-12.0
321	-4.0	-12.0
322	5.0	-5.0
323	8.0	0.0
324	5.0	-3.0
325	20.0	-1.0
326	20.0	4.0
327	6.0	0.0
328	13.0	-2.0
329	23.0	11.0
330	23.0	4.0
331	15.0	0.0
401	25.0	3.0
402	13.0	0.0
403	16.0	2.0
404	10.0	4.0
405	6.0	3.0
406	18.0	5.0
407	14.0	6.0
408	8.0	3.0
409	4.0	0.0

410	3.0	-2.0
411	3.0	-2.0
412	4.0	-2.0
413	12.0	-3.0
414	15.0	6.0
415	11.0	4.0
416	6.0	4.0
417	15.0	5.0
418	17.0	4.0
419	22.0	5.0
420	13.0	7.0
421	12.0	-1.0
422	0.0	-3.0
423	11.0	-5.0
424	16.0	-3.0
425	23.0	3.0
426	25.0	6.0
427	25.0	8.0
428	28.0	9.0
429	14.0	9.0
430	17.0	2.0
501	19.0	10.0
502	9.0	0.0
503	10.0	-3.0
504	16.0	-2.0
505	24.0	11.0
506	22.0	15.0
507	21.0	13.0
508	17.0	5.0
509	15.0	7.0
510	19.0	5.0
511	21.0	7.0
512	18.0	7.0
513	21.0	7.0
514	19.0	10.0
515	22.0	10.0
516	21.0	14.0
517	22.0	13.0
518	26.0	14.0
519	14.0	9.0
520	12.0	9.0
521	13.0	9.0
522	14.0	8.0
523	14.0	10.0
524	19.0	9.0
525	22.0	10.0
526	26.0	14.0
527	20.0	14.0
528	29.0	12.0
529	30.0	12.0
530	29.0	17.0

531	26.0	13.0
601	26.0	8.0
602	15.0	5.0
603	18.0	4.0
604	25.0	9.0
605	20.0	11.0
606	17.0	12.0
607	23.0	12.0
608	24.0	9.0
609	19.0	8.0
610	24.0	9.0
611	23.0	16.0
612	25.0	14.0
613	22.0	11.0
614	23.0	11.0
615	23.0	15.0
616	28.0	10.0
617	18.0	7.0
618	21.0	8.0
619	21.0	11.0
620	23.0	12.0
621	24.0	10.0
622	30.0	15.0
623	26.0	16.0
624	19.0	7.0
625	21.0	7.0
626	24.0	10.0
627	26.0	18.0
628	27.0	17.0
629	26.0	11.0
630	21.0	9.0
701	23.0	11.0
702	24.0	13.0
703	[illegible]	8.0
704	24.0	9.0
705	29.0	19.0
706	31.0	19.0
707	33.0	16.0
708	25.0	17.0
709	27.0	13.0
710	24.0	11.0
711	18.0	15.0
712	19.0	15.0
713	28.0	15.0
714	24.0	13.0
715	25.0	11.0
716	24.0	16.0
717	28.0	16.0
718	33.0	20.0
719	29.0	22.0
720	29.0	17.0

721	29.0	15.0
722	28.0	11.0
723	30.0	13.0
724	27.0	15.0
725	30.0	20.0
726	28.0	21.0
727	28.0	18.0
728	26.0	18.0
729	28.0	18.0
730	27.0	17.0
731	26.0	15.0
801	24.0	17.0
802	25.0	14.0
803	23.0	14.0
804	24.0	13.0
805	26.0	17.0
806	26.0	16.0
807	22.0	17.0
808	25.0	17.0
809	24.0	17.0
810	23.0	15.0
811	21.0	10.0
812	21.0	12.0
813	24.0	11.0
814	24.0	15.0
815	28.0	21.0
816	27.0	20.0
817	28.0	19.0
818	27.0	16.0
819	25.0	16.0
820	27.0	15.0
821	24.0	18.0
822	23.0	14.0
823	25.0	13.0
824	21.0	8.0
825	22.0	7.0
826	25.0	14.0
827	21.0	6.0
828	17.0	4.0
829	20.0	3.0
830	20.0	7.0
831	22.0	10.0
901	23.5	11.5
902	19.5	13.5
903	28.5	11.0
904	22.5	18.5
905	22.5	17.5
906	21.5	17.5
907	27.5	19.5
908	26.0	20.5
909	21.0	18.0

910	16.0	12.5
911	13.5	11.5
912	12.5	6.5
913	14.5	10.0
914	18.0	5.0
915	18.0	11.0
916	20.5	7.5
917	23.5	6.5
918	24.5	14.5
919	29.5	15.5
920	27.0	20.0
921	15.0	13.0
922	26.0	12.5
923	27.0	19.0
924	16.5	12.0
925	15.0	6.0
926	19.5	6.5
927	14.0	10.0
928	20.5	10.0
929	23.0	12.0
930	21.5	9.0
1001	13.5	10.0
1002	13.0	4.0
1003	16.5	4.0
1004	18.0	5.0
1005	13.0	9.0
1006	11.0	5.5
1007	20.5	11.0
1008	22.5	12.5
1009	25.0	13.5
1010	19.5	14.5
1011	20.0	10.0
1012	22.5	10.5
1013	29.5	13.0
1014	28.5	20.0
1015	25.5	16.5
1016	19.5	14.0
1017	23.5	10.5
1018	25.5	10.5
1019	26.0	18.0
1020	25.0	17.0
1021	29.5	14.5
1022	19.5	12.0
1023	20.5	10.5
1024	28.5	10.5
1025	32.0	21.0
1026	23.0	17.0
1027	20.0	14.5
1028	28.0	9.0
1029	26.5	20.0
1030	20.0	14.5

1031	19.0	12.0
1101	21.5	8.5
1102	21.5	9.0
1103	23.0	9.0
1104	25.5	10.0
1105	29.0	15.0
1106	26.0	18.0
1107	28.0	19.5
1108	28.0	17.0
1109	28.5	14.0
1110	31.0	16.0
1111	18.0	15.0
1112	21.5	8.5
1113	29.5	15.0
1114	30.5	19.5
1115	19.5	18.0
1116	25.0	15.0
1117	25.5	11.5
1118	25.5	17.0
1119	20.0	13.0
1120	19.0	13.0
1121	20.0	12.0
1122	20.5	9.5
1123	21.5	8.0
1124	17.0	12.5
1125	17.5	15.5
1126	20.0	16.5
1127	24.0	16.5
1128	22.0	18.0
1129	18.0	13.0
1130	15.0	13.5
1201	23.5	11.5
1202	19.5	13.5
1203	28.5	11.0
1204	22.5	18.5
1205	22.5	17.5
1206	21.5	17.5
1207	27.5	19.5
1208	26.0	20.5
1209	21.0	18.0
1210	16.0	12.5
1211	13.5	11.5
1212	12.5	6.5
1213	14.5	10.0
1214	18.0	5.0
1215	18.0	11.0
1216	20.5	7.5
1217	23.5	6.5
1218	24.5	14.5
1219	29.5	15.5
1220	27.0	20.0

```
1221  15.0  13.0
1222  26.0  12.5
1223  27.0  19.0
1224  16.5  12.0
1225  15.0   6.0
1226  19.5   6.5
1227  14.0  10.0
1228  20.5  10.0
1229  23.0  12.0
1230  21.5   9.0
1231  21.5   9.0

WEATHER.FLG
-----------

 101 N
 102 N
 103 N
 104 N
 105 N
 106 N
 107 N
 108 N
 109 N
 110 N
 111 N
 112 N
 113 N
 114 N
 115 N
 116 N
 117 N
 118 N
 119 N
 120 N
 121 N
 122 N
 123 N
 124 N
 125 N
 126 N
 127 N
 128 N
 129 N
 130 N
 131 N
 201 N
 202 N
 203 N
 204 N
```

205	N
206	N
207	N
208	N
209	N
210	N
211	N
212	N
213	N
214	N
215	N
216	N
217	N
218	N
219	N
220	N
221	N
222	N
223	N
224	N
225	N
226	N
227	N
228	N
229	N
301	N
302	N
303	N
304	N
305	N
306	N
307	N
308	N
309	N
310	N
311	N
312	N
313	N
314	N
315	N
316	N
317	N
318	N
319	N
320	N
321	N
322	N
323	N
324	N
325	N
326	N

327	N
328	N
329	N
330	N
331	N
401	N
402	N
403	N
404	N
405	N
406	N
407	N
408	N
409	N
410	N
411	N
412	N
413	N
414	N
415	N
416	N
417	N
418	N
419	N
420	N
421	N
422	N
423	N
424	N
425	N
426	N
427	N
428	N
429	N
430	N
501	N
502	N
503	N
504	N
505	N
506	N
507	N
508	N
509	N
510	N
511	N
512	N
513	N
514	N
515	N
516	N

517	N
518	N
519	N
520	N
521	N
522	N
523	N
524	N
525	N
526	N
527	N
528	N
529	N
530	N
531	N
601	N
602	N
603	N
604	N
605	N
606	N
607	N
608	N
609	N
610	N
611	N
612	N
613	N
614	N
615	N
616	N
617	N
618	N
619	N
620	N
621	N
622	N
623	N
624	N
625	N
626	N
627	N
628	N
629	N
630	N
701	N
702	N
703	N
704	N
705	N
706	N

707	N
708	N
709	N
710	N
711	N
712	N
713	N
714	N
715	N
716	N
717	N
718	N
719	N
720	N
721	N
722	N
723	N
724	N
725	N
726	N
727	N
728	N
729	N
730	N
731	N
801	N
802	N
803	N
804	N
805	N
806	N
807	N
808	N
809	N
810	N
811	N
812	N
813	N
814	N
815	N
816	N
817	N
818	N
819	N
820	N
821	N
822	N
823	N
824	N
825	N
826	N

827	N
828	N
829	N
830	N
831	N
901	N
902	N
903	N
904	N
905	N
906	N
907	N
908	N
909	N
910	N
911	N
912	N
913	N
914	N
915	N
916	N
917	N
918	N
919	N
920	N
921	N
922	N
923	N
924	N
925	N
926	N
927	N
928	N
929	N
930	N
1001	N
1002	N
1003	N
1004	N
1005	N
1006	N
1007	N
1008	N
1009	N
1010	N
1011	N
1012	N
1013	N
1014	N
1015	N
1016	N

1017	N
1018	N
1019	N
1020	N
1021	N
1022	N
1023	N
1024	N
1025	N
1026	N
1027	N
1028	N
1029	N
1030	N
1031	N
1101	N
1102	N
1103	N
1104	N
1105	N
1106	N
1107	N
1108	N
1109	N
1110	N
1111	N
1112	N
1113	N
1114	N
1115	N
1116	N
1117	N
1118	N
1119	N
1120	N
1121	N
1122	N
1123	N
1124	N
1125	N
1126	N
1127	N
1128	N
1129	N
1130	N
1201	N
1202	N
1203	N
1204	N
1205	N
1206	N

```
1207 N
1208 N
1209 N
1210 N
1211 N
1212 N
1213 N
1214 N
1215 N
1216 N
1217 N
1218 N
1219 N
1220 N
1221 N
1222 N
1223 N
1224 N
1225 N
1226 N
1227 N
1228 N
1229 N
1230 N
1231 N

WEATHER.INF
-----------

 366
```

MASTER.87

Essex Standard Weather	ESSEXSW	Essex	ESSEXSW	T	M	M	M	M	F	F	F	M
Welacky	WELACKY	Essex	ESSEXSW	F	T	F	T	F	T	F	T	T
Wright	WRIGHT	Essex	WRIGHT	T	B	M	B	M	T	F	T	B
Murray	MURRAY	Essex	ESSEXSW	F	T	F	T	F	T	F	T	T
Medel	MEDEL	Essex	MEDEL	T	B	M	B	M	T	F	T	B
Neal	NEAL	Essex	ESSEXSW	F	T	F	T	F	T	F	T	T
Nickel	NICKEL	Essex	ESSEXSW	F	T	F	T	F	T	F	T	T
Delange	DELANGE	Essex	ESSEXSW	F	T	F	T	F	T	F	T	T
Eastman	EASTMAN	Essex	EASTMAN	T	B	M	B	M	T	F	T	B
McGuigan	MCGUIGAN	Essex	ESSEXSW	F	T	F	T	F	T	F	T	T
Middlesex Standard Weather	MIDDLESW	Middlesex	MIDDLESW	T	M	M	M	M	F	F	F	M
VanBree	VANBREE	Middlesex	VANBREE	T	B	M	B	M	T	F	T	B
VanDiepen	VANDIEPE	Middlesex	VANDIEPE	T	B	M	B	M	T	F	T	B
Veens	VEENS	Middlesex	MIDDLESW	F	T	F	T	F	T	F	T	T
Woodlynn	WOODLYNN	Middlesex	WOODLYNN	T	B	M	B	M	T	F	T	B
Marsh	MARSH	Middlesex	MIDDLESW	F	T	F	T	F	T	F	T	T
Somers	SOMERS	Oxford	SOMERS	T	B	M	B	M	T	F	T	B
Smith	SMITH	Oxford	SMITH	T	B	M	B	M	T	F	T	B
Mulholland	MULHOLLA	Oxford	SOMERS	F	T	F	T	F	T	F	T	T
Rokeby	ROKEBY	Oxford	ROKEBY	T	B	M	B	M	T	F	T	B
Thomas	THOMAS	Oxford	SOMERS	F	T	F	T	F	T	F	T	T
Norfolk Standard Weather	NORFOLSW	Norfolk	NORFOLSW	T	M	M	M	M	F	F	F	M
Lachapelle	LACHAPEL	Norfolk	LACHAPEL	T	B	M	B	M	T	F	T	B
Vittoria	VITTORIA	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
Kerry	KERRY	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
Cooper	COOPER	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
Totzke	TOTZKE	Norfolk	TOTZKE	T	B	M	B	M	T	F	T	B
Ryder	RYDER	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
Cleaver	CLEAVER	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
H.E.S.	HES	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
Brant	BRANT	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
VanYmeren	VANYMERE	Norfolk	NORFOLSW	F	T	F	T	F	T	F	T	T
Waters	WATERS	Norfolk	WATERS	T	B	M	B	M	T	F	T	B
Vineland Standard Weather	VINELASW	Vineland	VINELASW	T	M	M	M	M	F	F	F	M
Trostenko	TROSTENK	Vineland	VINELASW	F	T	F	T	F	T	F	T	T
Jahnke	JAHNKE	Vineland	VINELASW	F	T	F	T	F	T	F	T	T
Retzik	RETZIK	Vineland	VINELASW	F	T	F	T	F	T	F	T	T
HRIO	HRIO	Vineland	VINELASW	F	T	F	T	F	T	F	T	T
Robertsonn	ROBSONN	Vineland	VINELASW	F	T	F	T	F	T	F	T	T
Overbeeke	OVERBEEK	Vineland	OVERBEEK	T	B	M	B	M	T	F	T	B
Bousfield	BOUSFIEL	Halton	BOUSFIEL	T	B	M	B	M	T	F	T	B
Cross	CROSS	Halton	BOUSFIEL	F	T	F	T	F	T	F	T	T
Pallett	PALLETT	Halton	PALLETT	T	B	M	B	M	T	F	T	B
Bennett	BENNETT	Halton	BENNETT	T	B	M	B	M	T	F	T	B
Sanderson	SANDERSO	Halton	BOUSFIEL	F	T	F	T	F	T	F	T	T
Stroz	STROZ	Halton	BOUSFIEL	F	T	F	T	F	T	F	T	T
Bowmanville Standard Weather	BOWMANSW	Bowmanville	BOWMANSW	T	M	M	M	M	F	F	F	M

```
Death                         DEATH    Bowmanville    DEATH    T B M B M T F T B
Vanderwindt                   VANDERWI Bowmanville    VANDERWI T B M B M T F T B
Archibald                     ARCHIBAL Bowmanville    BOWMANSW F T F T F T F T T
Gibson                        GIBSON   Bowmanville    BOWMANSW F T F T F T F T T
Carruthers                    CARRUTHE Bowmanville    CARRUTHE T B M B M T F T B
Jose                          JOSE     Bowmanville    BOWMANSW F T F T F T F T T
Highland                      HIGHLAND Bowmanville    BOWMANSW F T F T F T F T T
Sikma                         SIKMA    Bowmanville    BOWMANSW F T F T F T F T T
Georgian Bay Standard Weather GEOBAYSW Georgian Bay   GEOBAYSW T M M M M F F F M
Meester                       MEESTER  Georgian Bay   MEESTER  T B M B M T F T B
Beckwith                      BECKWITH Georgian Bay   GEOBAYSW F T F T F T F T T
Leuty                         LEUTY    Georgian Bay   LEUTY    T B M B M T F T B
Comly                         COMLY    Georgian Bay   GEOBAYSW F T F T F T F T T
Ardiel                        ARDIEL   Georgian Bay   GEOBAYSW F T F T F T F T T
Winters                       WINTERS  Georgian Bay   WINTERS  T B M B M T F T B
Douglen                       DOUGLEN  Georgian Bay   GEOBAYSW F T F T F T F T T
Kemptville Standard Weather   KEMPTVSW Kemptville     KEMPTVSW T M M M M F F F M
Hall                          HALL     Kemptville     KEMPTVSW F T F T F T F T T
Dentz                         DENTZ    Kemptville     DENTZ    T B M B M T F T B
Hobson                        HOBSON   Kemptville     KEMPTVSW F T F T F T F T T
Robertson                     ROBSON   Kemptville     KEMPTVSW F T F T F T F T T
Smyth                         SMYTH    Kemptville     KEMPTVSW F T F T F T F T T
Quinte Standard Weather       QUINTESW Quinte         QUINTESW T M M M M F F F M
Knight                        KNIGHT   Quinte         KNIGHT   T B M B M T F T B
Appleby                       APPLEBY  Quinte         QUINTESW F T F T F T F T T
Zimmermin                     ZIMMERMI Quinte         QUINTESW F T F T F T F T T
Dunnett                       DUNNETT  Quinte         DUNNETT  T B M B M T F T B
S.E.S.                        SES      Quinte         QUINTESW F T F T F T F T T
Reid                          REID     Quinte         REID     T B M B M T F T B
Bonter                        BONTER   Quinte         BONTER   T B M B M T F T B
Rutter                        RUTTER   Quinte         RUTTER   T B M B M T F T B
Hughes                        HUGHES   Quinte         QUINTESW F T F T F T F T T
Kelly                         KELLY    Quinte         KELLY    T B M B M T F T B
Gates                         GATES    Quinte         QUINTESW F T F T F T F T T
```

MASTER.DIR

This file is a copy of MASTER.87.

APPENDIX 3. BUGWATCH PROGRAM FILES

```
BUGWATCH.PAS
------------

Program Pest_Monitoring_System( Input, Output );
{
  Authors    : John Yee
               Research Station
               Agriculture Canada
               Vineland Station, Ontario L0R 2E0

               Sandra Yee
               Dept. Mathematics
               Brock University
               St. Catharines, Ontario

  Date       : September 1988

  Version    : 2.01

  Purpose    : 1. Calculate insect lifestage development up to the current date
                  and predict development for days on which forecast temperatures
                  have been entered.

               2. Extract and print a summary of information from insect trap
                  catch data.

               3. Extract and print summary reports for a region.

               4. Extract and plot a graph of average trap count versus day.  Plot
                  may be printed on paper at the user's discretion.

  Comments   : 1. This program was written for the Ontario Ministry of Agriculture
                  and Food ( OMAF ). The program is based on an Agriculture Canada
                  system originally written in BASIC and run on a Digital Equipment
                  Corporation VAX operating under VMS.

               2. This program was written in Turbo Pascal 3.0 ( Borland[ 1985 ] )
                  for an IBM-PC/XT operating under PC-DOS 2.10.  The program
                  does not use full screen editing; it should run on an IBM-PC,
                  PC/XT, PC/AT or compatible computer.

               3. The entire Pest Monitoring System is stored in the directory
                  C:\OMAF-A-C and subdirectories.

               4. A leap year ( 366 days ) is used for all date calculations.
  References :

             Borland International Inc., 1985.
                  Turbo Pascal version 3.0 Reference Manual
                  4585 Scotts Valley Drive
                  Scotts Valley, CA, USA  95066


  Change Log : Version 1.0, April 1987
                  original version released to OMAF

             Version 1.1, July 1987
                  more in-line commentary
                  minor changes to improve robustness to errors
                  minor algorithm changes

                  known bugs . BUGWATCH.PAS procedure valid_weather_site
                             . INP-TRAP.PAS procedure next_trap_site

             Version 2.0, May 1988
                  added graphics - plot of trap count versus day
                  added regional report section
}

const
    beep      = #07; { ASCII BEL }
    form_feed = #12;

    directory_size       = 200;            { maximum number of monitoring sites;
                                             this value is arbitrary, there is no theoretical limit }
    end_directory_string = 'end dir';      { last record in the master directory disk file }
    PMS_root_directory   = 'C:\OMAF-A-C\';
    parameter_file_name  = 'BUGSYS.PAR';   { file containing BUGWATCH program parameters }

    nomore = 'NOMORE';

    grand_maximum_traps = 10; { maximum number of traps used ( maximum taken over all pests ) }

    width_of_site_name_field    =  8; { in MASTER directory }
    width_of_full_name_field    = 30;
    width_of_region_field       = 15;

    width_of_date_field         =  4; { date as MMDD }
    width_of_temperature_field  =  6; { temperature as a real number }

    { widths of the fields in the rate parameter file *.RAT }
    width_of_rate_index_field   =  3; { temperature as an integer }
    width_of_rate_field         =  9; { rate as a real number     }

    { widths of the fields in the lifestage development file *.MDL }
    width_of_stage_name_field   = 30;
    { widths of the fields in the trap catch file *.TRP }
    width_of_average_field      =  7; { average per trap }
    width_of_todays_total_field =  6; { total count for this date }
    width_of_ntraps_field       =  3; { number of traps for this date }
    width_of_grand_total_field  =  8; { grand total count to this date }
    width_of_stats_field        = 24; { width_of_average + width_of_todays_total +
                                        width_of_ntraps_field + width_of_grand_total }
    width_of_trap_field         =  7; { trap count }

type
    numeric_date_string_type = string[ width_of_date_field ];

    disk_file_name      = string[ width_of_site_name_field ];
    full_site_name_type = string[ width_of_full_name_field ];
    region_name_type    = string[ width_of_region_field ];

    line                = string[ 80 ];

    kinds_of_sites = ( valid_site, weather, first_pest, apple_maggot, apanteles_ornigis, codling_moth,
                       european_red_mite, oblique_banded_leaf_roller, oriental_fruit_moth,
                       red_banded_leaf_roller, tentiform_leaf_miner, last_pest );
                     {
                       "First_pest" and "last_pest" are dummy types; they are used to facilitate
                       entering or deleting pest sites at the ends of the pest list.  Loop constructions
                       are easier to implement by initializing the loop variable to succ( first_pest )
                       rather than naming explicitly the first pest in the list.
                     }

    directory_entry = record
                           full_name                : full_site_name_type;
                           region                   : region_name_type;
                           site_name,
                                nearest_weather_site : disk_file_name;
                                                     {
                                                       Not all sites log weather data; these sites
                                                       use weather data from another site.
                                                     }

                           trapped, modelled        : array[ weather..last_pest ] of boolean;
                                                     {
                                                       These vectors indicate whether or not the
                                                       pest is trapped or modelled at this site.
                                                       Example : trapped[ apple_maggot ] := true;
                                                                 modelled[ apple_maggot ] := false;

                                                                 means apple maggot is trapped,
                                                                 but is not modelled at this site
                                                     }
                      end; { record }

const
    max_pest_stages = 8; { maximum number of pest lifestages modelled }
    pest_name_vector : array[ first_pest..last_pest ] of line =
                       ( 'FIRST PEST', 'APPLE MAGGOT', 'APANTELES ORNIGIS', 'CODLING MOTH', 'EUROPEAN RED MITE',
                         'OBLIQUE BANDED LEAF ROLLER', 'ORIENTAL FRUIT MOTH', 'RED BANDED LEAF ROLLER',
                         'TENTIFORM LEAF MINER', 'LAST PEST' );

    pest_name_vector_abbr : array[ first_pest..last_pest ] of line =
                            ( 'FIRST PEST', 'AM', 'AO', 'CM', 'ERM', 'OBLR', 'OFM', 'RBLR', 'TLM', 'LAST PEST' );

    pest_file_name_vector : array[ first_pest..last_pest ] of disk_file_name =
                            ( 'FPEST', 'AMAGGOT', 'AORNIGIS', 'CODLING', 'ERM', 'OBLR', 'OFMOTH', 'RBLR', 'STLM', 'LPEST' );
                            {
                              File naming convention for disk files.
                            }

var
    command   : char; { The primary tasks which the system will perform are identified by a single character. }
    task_code : line;

    i,
    this_year, todays_Julian_date : integer;
    Year_path, Year_subdirectory  : line;

    max_master_directory_index : integer;
    master_directory           : array[ 1.. directory_size ] of directory_entry;
    pest_trapped_at_some_site  : array[ first_pest..last_pest ] of boolean;

    { BUGWATCH parameters; these are read from BUGSYS.PAR }
    width_of_paper          : real;    { width of printer paper in inches from edge to edge }
    characters_per_inch     : integer; { CPI for regional compilation }
    printer_init_sequence   : line;    { printer initialization sequence }
    printer_term_sequence   : line;    { printer termination sequence }
    screen_color_code       : integer; { HiRes graphics screen color }
    trap_plot_x_axis        : array[ first_pest .. last_pest, 1..2 ] of integer; { minimum and maximum Julian date }

{ command line parameters }
const
    max_command_line_parameters = 5;
var
    n_command_line_parameters : integer;
    command_line_parameter    : array[ 1..max_command_line_parameters ] of line;
    one_parameter             : line;
    val_result                : integer;


function read_response( prompt_string : line ) : line;
    {
      This function acts like a prompting read statement.  A prompt string is
      written on the screen and keyboard input is read.  The function then returns
      the input string.
    }
    var
```

```
        response : line;
    begin

        write( prompt_string );
        readln( response );
        read_response := response;

    end; { function read_response }


procedure response_unrecognized( var response : line );
    {
      This procedure informs the user that the input is unrecognized by the program.
    }
    begin

        writeln;
        writeln( beep + beep + beep + ' The response "', response, '" is unrecognized.' );
        writeln;

    end; { procedure response_unrecognized }


function file_exist( file_name : line ) : boolean;
    {
      This function determines whether or not a file exits on disk.
      If the file exits, the function returns TRUE, otherwise FALSE.
      ref : Borland [ 1985 ] p. 97
    }
    var
        file_var : file; { use untyped file }
    begin

        assign( file_var, file_name );
        {$I-}
        reset( file_var );
        {$I+}

        file_exist := ( IOresult = 0 );
        close( file_var );

    end; { function file_exist }

{$I str_proc.pas }  { string manipulating procedures and functions }


{$I siteproc.pas }  { procedures and functions for accessing, searching, etc. the master directory }


{$I dateproc.pas }  { procedures and functions for manipulating calendar dates }
```

```
procedure read_BUGWATCH_parameters;
    {
      This procedure reads the BUGWATCH program parameters from BUGSYS.PAR.
    }
    var
        file_var       : text;
        parameter_file : line;
        i              : integer;
        pest           : kinds_of_sites;
        date1, date2   : string[ width_of_date_field ];
        space          : char;
    begin

        parameter_file := PMS_root_directory + parameter_file_name;
        assign( file_var, parameter_file );
        reset( file_var );

        readln( file_var, width_of_paper );
        readln( file_var, characters_per_inch );

        printer_init_sequence := '';
        read( file_var, i );
        repeat
             printer_init_sequence := printer_init_sequence + chr( i );
             read( file_var, i );
        until ( i < 0 );
        readln( file_var );

        printer_term_sequence := '';
        read( file_var, i );
        repeat
             printer_term_sequence := printer_term_sequence + chr( i );
             read( file_var, i );
        until ( i < 0 );
        readln( file_var );

        readln( file_var, screen_color_code );

        for pest := succ( first_pest ) to pred( last_pest ) do
             begin
                  readln( file_var, date1, space, date2 );
                  trap_plot_x_axis[ pest, 1 ] := numeric_date_to_Julian( date1 );
                  trap_plot_x_axis[ pest, 2 ] := numeric_date_to_Julian( date2 );
             end; { for pest }

        close( file_var );

    end; { procedure read_BUGWATCH_parameters }

procedure welcome;
```

```
    {
      This procedure displays an initial message on the screen.
    }
    begin

        ClrScr;
        GotoXY( 1, 10 );
        writeln( '                    Ontario Ministry of Agriculture and Food' );
        writeln( '                                       and' );
        writeln( '                               Agriculture Canada' );
        writeln;
        writeln( '                             Pest Monitoring System' );
        writeln;
        delay( 4000 ); { 4000 millisecond delay }

    end; { procedure welcome }


procedure display_command_menu;
    {
      This procedure displays the primary list of tasks BUGWATCH will perform
      ( the main menu ).  The code letters are the input characters by which
      the user selects a task.
    }
    begin

        ClrScr;
        writeln;
        writeln;
        writeln( '       List of Tasks which the system will perform' );
        writeln;
        writeln( '       Code    Task' );
        writeln( '       ====    ====' );
        writeln;
        writeln( '         W     input Weather data' );
        writeln;
        writeln( '         T     input Trap catch data' );
        writeln;
        writeln( '         C     Calculate pest development and trap catch summaries' );
        writeln;
        writeln( '         P     Print reports' );
        writeln;
        writeln( '         R     print Regional reports' );
        writeln;
        writeln( '         G     plot trap catch Graphs' );
        writeln;
        writeln( '         E     Exit the Pest Monitoring System' );
        writeln;

    end; { procedure display_command_menu }
```

```
procedure sign_off;
    {
      This procedure displays a sign-off message after an Exit command is
      issued.  It provides the user with some feedback that the program has
      completed execution normally.
    }
    begin

        writeln;
        writeln( 'End of Pest Monitoring System.' );
        writeln;

    end; { procedure sign_off }


{$I inp-wthr.pas }  { input weather data }


{$I inp-trap.pas }  { input trap count data }


{$I calculat.pas }  { calculate lifestage development and trap count summaries }


{$I printrep.pas }  { print reports }


{$I regional.pas }  { print regional reports }


{$I graftrap.pas }  { graph average trap catches }


{                   Main   Program                   }

begin

    Julian_date_from_DOS( this_year, todays_Julian_date );

    { obtain command line parameters }
    n_command_line_parameters := ParamCount;
    if   n_command_line_parameters > max_command_line_parameters
    then n_command_line_parameters := max_command_line_parameters;
    for i := 1 to n_command_line_parameters do command_line_parameter[ i ] := ParamStr( i );

    if   n_command_line_parameters > 0
    then
         begin
            { current version fetches only the first command line parameter }
            one_parameter := upper_case( command_line_parameter[ 1 ] );
            if   Pos( 'YEAR', one_parameter ) = 1
            then
```

```
              begin
                  delete( one_parameter, 1, 5 ); { delete 'YEAR=' from the string }
                  val( one_parameter, i, val_result );
                  if   val_result = 0
                  then this_year := 1
                  else
                       begin
                           { extract year field }
                           one_parameter := command_line_parameter[ 1 ];
                           delete( one_parameter, 1, 5 );

                           writeln( beep + 'Cannot process "' + one_parameter + '" as a valid year.' );
                           writeln( 'Assuming YEAR=', this_year : 4 );
                           delay( 5000 );
                       end; { if val_result = 0 }
              end
         else
              begin
                  writeln( beep + 'Cannot process YEAR option "' + command_line_parameter[ 1 ], '"' );
                  writeln( 'Assuming YEAR=', this_year : 4 );
                  delay( 5000 );
              end; { if Pos( 'YEAR', one_parameter ) = 1 }
     end; { if n_command_line_parameters > 0 }

{ construct name of year subdirectory }
str( this_year : 4, Year_subdirectory );
Year_path := 'YEAR'+ Year_subdirectory + '\';

welcome;

read_BUGWATCH_parameters;
fetch_master_directory;
fetch_pest_trapped_at_some_site;

repeat

     display_command_menu;

     task_code := read_response( ' Enter the code of the task : ' );
     command   := extract_alphabetic_char( task_code );
     command   := upper_case( command );

     case command of
          'W' : input_weather;
          'T' : input_trap_catch;
          'C' : calculations;
          'P' : print_report;
          'R' : print_regional_reports;
          'G' : graph_trap_catch;
          'E' : { null line to save a place for the EXIT command };
          else  begin
                    response_unrecognized( task_code );
                    end;
         end; { case command }
    until ( command = 'E' ); { E for EXIT }
    sign_off;
end. { Program Pest_Monitoring_System }
```

STR_PROC.PAS

```
function delete_blanks( initial_string : line ) : line;
    {
      This function returns a string with all spaces deleted from the
      initial string.
    }
    var
        i, len : integer;
        dummy  : line;
        ch     : char;
    begin

        dummy := '';
        len   := length( initial_string );

        i := 1;
        while ( i <= len ) do
            begin
                ch := initial_string[ i ];
                if   ch <> ' '
                then dummy := dummy + ch;
                i := i + 1;
            end; { while }

        delete_blanks := dummy;

    end; { function delete_blanks }


function upper_case( initial_string : line ) : line;
    {
      This function converts all lower case alphabetic characters
      to their upper case equivalents.  Non-lower case alphabetic
      characters are left unchanged.

      Reference : Turbo Pascal Manual
                  p. 148
    }
    var
        i      : integer;
        dummy  : line;
    begin

        dummy := initial_string;

        for i := 1 to Length( initial_string ) do
            dummy[ i ] := UpCase( initial_string[ i ] );

        upper_case := dummy;
    end; { function upper_case }

function last_characters_in_string( initial_string : line; number_of_characters : integer ) : line;
    {
      This function returns the last "number_of_characters" characters from the
      initial string.  If "number_of_characters" exceeds the length of the initial
      string, then the entire string is returned; no pad characters are inserted.
    }
    var
        start_col, len : integer;
        dummy  : line;
    begin

        len := length( initial_string );
        if   number_of_characters < len
        then
             begin
                 start_col := len - number_of_characters + 1;
                 dummy := copy( initial_string, start_col, number_of_characters );
             end { if }
        else
             dummy := initial_string;

        last_characters_in_string := dummy;

    end; { function last_characters_in_string }


function extract_alphabetic_char( var response : line ) : char;
    {
      This function searches for the first alphabetic character from the string
      variable "response".

      If the search succeeds, the function returns the character.
      If the search fails, the function returns the last character in the string.
      If the input string is empty, a space is returned.
    }
    var
        ch     : char;
        i, len : integer;

    begin

        len := length( response );

        if   len > 0
        then
             begin
                 i   := 1;
                 repeat
                      ch := response[ i ];
```

```
                  i := i + 1;
             until ( ( ch in  [ 'A'..'Z', 'a'..'z' ] ) or ( i > len ) );
             extract_alphabetic_char := ch;
         end
    else
         extract_alphabetic_char := ' ';

end; { function extract_alphabetic_char }
```

```
SITEPROC.PAS
------------

function found_in_directory( site_name : disk_file_name; site_directory_address : integer;
                             site_type : kinds_of_sites ) : boolean;
   {
    This function searches the master directory for the site specified by "site_name"
    and determines whether or not the site collects ( traps ) the data specified by
    "site_type".  The function uses "site_directory_address" and "site_name" in a simple
    error check.

    If the site traps the data, the function returns TRUE.  If the site does not trap the
    data or the simple error check returns FALSE, the function returns FALSE.

    Note : Weather data are 'trapped' at weather sites.  Hence the following example.

    Example : found_in_directory( 'ESSEXSW', essexsw_address, weather ) returns the
              value TRUE.
   }
   var
      check : boolean;
   begin

      { make sure the site_name and site_directory_address correspond
        to the same site in the master directory }
      check := ( site_name = master_directory[ site_directory_address ].site_name );

      if   check
      then found_in_directory := master_directory[ site_directory_address ].trapped[ site_type ]
      else found_in_directory := false;

   end; { function found_in_directory }


procedure search_for_site_in_directory( var site_name : disk_file_name; var full_site_name : full_site_name_type;
                                        var site_exists : boolean; var directory_address : integer );
   {
    This procedure searches the master directory for the site specified by
    "site_name" or "full_site_name".  The search stops at the first entry
    which matches "site_name" or "full_site_name" or at the end of the master
    directory.

    If the search succeeds, the procedure returns the "site_name",
    "full_site_name" and "directory_address" of the master directory entry;
    the boolean variable "site_exists" is returned TRUE.  If the search fails,
    the procedure returns "site_name" and "full_site_name" unchanged,
    "directory_address" is set to 0, and "site_exists" returns FALSE.
   }
   var
      i                                : integer;
      item1, item2, squished_full_name : line;
      found                            : boolean;
    begin

        squished_full_name := upper_case( delete_blanks( full_site_name ) );

        i := 1;
        repeat
             item1 := master_directory[ i ].site_name;

             { 'squish' ( delete spaces ) and capitalize the master directory full name to
                ensure that it will be in exactly the same format as "squished_full_name" }
             item2 := upper_case( delete_blanks( master_directory[ i ].full_name ) );

             found := ( item1 = site_name ) or ( item2 = squished_full_name );
             i := i + 1;
        until ( found or ( item1 = end_directory_string ) );

        if   found
        then
             begin
                 directory_address := i - 1;
                 site_name := master_directory[ directory_address ].site_name;
                 full_site_name := master_directory[ directory_address ].full_name;
             end
        else
             directory_address := 0;

        site_exists := found;

    end; { procedure site_exists }

procedure increment_to_next_site( var site_name : disk_file_name; var full_site_name : full_site_name_type;
                                  var site_exists : boolean; var site_directory_address : integer );
    {
      This procedure increments the site_directory_address to the next site.  The "site_name" and
      "full_site_name" strings are reset to correspond to the new site, and the
      boolean flag "site_exists" is set to TRUE.

      If the current site is the last one, then the procedure returns to the
      first one.
    }
    begin

        repeat

             site_directory_address := ( site_directory_address + 1 ) MOD ( directory_size + 1 );
             {
               Note : The MOD divisor " directory_size + 1 " is used to avoid a
                      quotient of 0 for site_directory_address = directory_size - 1.
             }

             site_name      := master_directory[ site_directory_address ].site_name;
```

```
          full_site_name := master_directory[ site_directory_address ].full_name;
          site_exists    := true;

          if   site_name = end_directory_string { if   at end of directory           }
          then site_directory_address := 0;      { then start at the beginning again }

       until ( site_name <> end_directory_string );

    end; { procedure increment_to_next_site }


procedure fetch_site_name( prompt : line; var response : line; var site_name : disk_file_name;
                           var full_site_name : full_site_name_type; var site_exists : boolean;
                           var site_directory_address : integer );
    {
      This procedure prompts the user for a site name.

      The prompt includes a default site name.  The user may enter an immediate
      <RETURN> or a site name.  If the input string is null, the procedure
      assumes the user wants the default site name, otherwise the procedure
      accepts the input string as the site name.

      The procedure searches for the requested site in master_directory.  If the
      site is found, the site parameters are reset to correspond to that site,
      otherwise the procedure alerts the user that the site cannot be found and
      reprompts for another site.

      The procedure terminates when the user has entered a site which is in
      master_directory or has entered "NOMORE".  The user's input is also
      returned in the string variable "response"; this is primarily to return
      the "NOMORE" response to the calling block.
    }
    var
       prompt_string, dummy_response : line;
       old_site_name                 : disk_file_name;
    begin

       old_site_name := site_name;

       repeat

          { default site always remains the same }
          prompt_string  := prompt + ' site ( ' + old_site_name + ' ) ? ';

          dummy_response := read_response( prompt_string );
          dummy_response := upper_case( delete_blanks( dummy_response ) );

          if   dummy_response <> ''
          then
               { If the input string is not null, then accept the input
                 string as the site name ( both short and long forms ). }
               begin
```

```
                      site_name      := dummy_response;
                      full_site_name := dummy_response;
                  end
             else
                  { If the input string is null, then assume the user wants the
                    default site. }
                  begin
                      site_name      := old_site_name;
                      full_site_name := old_site_name;
                  end;

             if   dummy_response <> nomore
             then
                  begin
                      search_for_site_in_directory( site_name, full_site_name, site_exists, site_directory_address );
                      if   not site_exists
                      then writeln( beep + ' Requested site cannot be found in the MASTER directory.' );
                  end; { if dummy_response }

        until ( site_exists or ( dummy_response = nomore ) );

        response := dummy_response;

    end; { procedure fetch_site_name }


procedure valid_weather_site( var site_name : disk_file_name; var full_site_name : full_site_name_type;
                              var site_exists : boolean; var site_directory_address : integer );
    {
      This procedure checks whether or not the site specified by "site_name"
      gathers weather data.  If the site does not gather weather data, the
      procedure searches for the next site which does and resets "site_name",
      "full_site_name", "site_directory_address", and "site_exists" accordingly.

      This procedure is similar to procedure next_trap_site.

      BUG : An erroneous weather site may be returned.  See Note following
            the "repeat-until" loop.
    }
    var
        original_site : disk_file_name;
        OK_site       : boolean;
    begin

        original_site := site_name;

        repeat

             OK_site := found_in_directory( site_name, site_directory_address, weather );
             if   not OK_site
             then increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );
    until ( OK_site or ( site_name = original_site ) );
    {
      Note : If the repeat loop gets back to the site first passed to the
             procedure, then there are no weather gathering sites in
             master_directory and the procedure must end.  If the
             stop condition ( site_name = original_site ) were not included,
             the repeat loop would execute indefinitely.

             In this case the procedure returns parameters which do not
             correspond to a site which gathers weather data.
    }

  end; { procedure valid_weather_site }


procedure fetch_master_directory;
  {
    This procedure reads the master directory disk file MASTER.DIR and stores
    the information in the program's internal master_directory array.  The
    information is converted into a form to match the master directory
    internal data format and type ( see declaration of type directory_entry ).

    The format of the master directory file MASTER.DIR is

    col. 0         1         2         3         4         5         6         7         8         9
         123456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789

         full site name            .sitename.region name    .wthrname. T F F F F F F F

    example :

         Site SANDRA test site      SANDRA   Test region      TESTSITE F F F F T F F F T


    The fields have the following meaning.

      field   meaning
      -----   -------

         1    full name of the site
         2    8 character abbreviation of the site name
         3    name of the region in which the site resides
         4    name of nearest weather site


              The code in the next set of fields indicates whether or not the
              site traps ( gathers ) and/or models data about the indicated
              entity.

                   code   meaning
                   ----   -------
```

```
                 T     TRAPPED only
                 M     MODELED only
                 B     both TRAPPED and MODELED
             other     neither TRAPPED nor MODELED

         5   weather
         6   apple maggot
         7   apanteles ornigis
         8   codling moth
         9   european red mite
        10   oblique banded leaf roller
        11   oriental fruit moth
        12   red banded leaf roller
        13   tentiform leaf miner

  Notes :
           1. For the insects, "TRAPPED" means trap counts are gathered.

           2. For the insects, "MODELLED" means lifestage development is
              modelled.

           3. For weather, "TRAPPED" means weather data are gathered.
              Currently, only maximum and minimum air temperatures are
              gathered.

           4. For weather, "MODELLED" means weather data are modelled.
              Currently, weather data are not modelled.

           5. The end-of-file marker for MASTER.DIR occurs at the end of
              the last line.
}
var
    i                         : integer;
    full_name                 : string[ 30 ];
    region_name               : string[ 16 ];
    site_name, weather_site   : string[ 9 ];
    code                      : string[ 2 ];
    T_flag                    : char;
    site_type                 : kinds_of_sites;
    directory_file            : text;
    directory_name            : line;

begin

    writeln;
    writeln( ' Fetching master directory.  Please wait.' );

    directory_name := PMS_root_directory + Year_path + 'MASTER.DIR';
    assign( directory_file, directory_name );
    reset( directory_file );

    i := 1;
```

```
while ( not eof( directory_file ) and ( i < directory_size ) ) do
    begin

        read( directory_file, full_name, site_name, region_name, weather_site );

        site_name    := delete_blanks( site_name );
        weather_site := delete_blanks( weather_site );

        master_directory[ i ].full_name := full_name;
        master_directory[ i ].site_name := site_name;
        master_directory[ i ].region    := last_characters_in_string( region_name, width_of_region_field );
        master_directory[ i ].nearest_weather_site :=
                                           last_characters_in_string( weather_site, width_of_site_name_field );

        { weather }
        read( directory_file, code );

        T_flag := upper_case( delete_blanks( code ) );
        master_directory[ i ].trapped[ weather ]  := ( T_flag in [ 'T', 'B' ] );
        master_directory[ i ].modelled[ weather ] := ( T_flag in [ 'M', 'B' ] );

        { pests }
        for site_type := succ( first_pest ) to pred( last_pest ) do
            begin

                read( directory_file, code );

                T_flag := upper_case( delete_blanks( code ) );
                master_directory[ i ].trapped[ site_type ]  := ( T_flag in [ 'T', 'B' ] );
                master_directory[ i ].modelled[ site_type ] := ( T_flag in [ 'M', 'B' ] );

            end; { for site_type }

        readln( directory_file );

        i := i + 1;

end; { while }

{ place end-of-directory marker string in the master directory array
  after the last entry from MASTER.DIR }
master_directory[ i ].full_name            := end_directory_string;
master_directory[ i ].site_name            := end_directory_string;
master_directory[ i ].region               := end_directory_string;
master_directory[ i ].nearest_weather_site := end_directory_string;

max_master_directory_index := i;

{ If the entire MASTER.DIR file can not be read into master_directory,
  then warn the user.  Increase the size of the master_directory array
  by increasing the value of the constant directory_size and then recompiling. }
if   not eof( directory_file )
```

```
        then
            begin
                writeln( beep + beep + beep );
                writeln( 'WARNING : The master directory file MASTER.DIR is too big.' );
                writeln( '          BUGWATCH will execute properly, but will not be able to' );
                writeln( '          access sites beyond site "', site_name, '".' );
                writeln;
            end;

        close( directory_file );

    end; { procedure fetch_master_directory }


function valid_trap_site( var site_name : disk_file_name; var site_directory_address : integer ) : boolean;
    {
      This function determines whether or not the site "site_name" gathers
      trap data for some pest.  If it does the function returns TRUE; if not
      the function returns FALSE.
    }
    var
        OK_site      : boolean;
        trapped_pest : kinds_of_sites;
    begin

        OK_site := false;
        trapped_pest := succ( first_pest );

        repeat
            OK_site := found_in_directory( site_name, site_directory_address, trapped_pest );
            trapped_pest := succ( trapped_pest );
        until ( OK_site or ( trapped_pest = last_pest ) );

        valid_trap_site := OK_site;

    end; { function valid_trap_site }


procedure next_trap_site( var site_name : disk_file_name; var site_directory_address : integer );
    {
      This procedure checks whether or not the site "site_name" gathers
      trap data on at least one pest.  If the site does not, then the
      procedure increments to the next site which does and returns the site
      name and directory address ( in master_directory ).

      This procedure is similar to procedure valid_weather_site.

      BUG : An erroneous trap site may be returned.  See Note following the
            "repeat-until" loop.
    }
    var
        OK_site         : boolean;
        original_site  : disk_file_name;
        full_site_name : full_site_name_type;
        site_exists    : boolean;
    begin

        original_site := site_name;

        repeat

             OK_site := valid_trap_site( site_name, site_directory_address );

             if   not OK_site
             then increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        until ( OK_site or ( site_name = original_site ) );
        {
          Note : If the repeat loop gets back to the site first passed to
                 the procedure, then there are no trap gathering sites in
                 master directory and the procedure must end.  If the stop
                 condition ( site_name = original_site ) were not included,
                 the repeat loop would execute indefinitely.

                 In this case the procedure returns parameters which do not
                 correspond to a site which gathers trap data.
        }

    end; { procedure next_trap_site }


procedure fetch_pest_trapped_at_some_site;
    {
      This procedure searches the master directory for at least one site which
      traps a given pest.  If such a site exists, the corresponding flag is
      set to TRUE, otherwise the flag is set to FALSE.
    }
    var
        one_trapped : boolean;
        i           : integer;
        pest        : kinds_of_sites;
    begin

        for pest := succ( first_pest ) to pred( last_pest ) do
            begin

                { search the master directory until a site is found which traps the
                  indicated pest or the end of the master directory is reached }
                i := 1;
                repeat
                     one_trapped := master_directory[ i ].trapped[ pest ];
                     i := i + 1;
                until ( ( i = max_master_directory_index ) or one_trapped );
            pest_trapped_at_some_site[ pest ] := one_trapped;
        end; { for pest }
end; { procedure fetch_pest_trapped_at_some_site }
```

```
DATEPROC.PAS
------------

function Julian_to_numeric_date( Julian_date : integer ) : numeric_date_string_type;
    {
      This function converts a Julian date to the format MMDD.  The Julian date
      is passed as an integer.  The numeric date MMDD is returned as a four
      character string.
      This function is the inverse of function numeric_date_to_Julian.
    }
    const
        end_of_month : array[ 0..13 ] of integer =
                         ( 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366, 397 );

    var
        month, day         : integer;
        month_str, day_str : numeric_date_string_type;

    begin

        { search for the month which ends after the Julian_date }
        month := 1;
        while  ( end_of_month[ month ] < Julian_date ) do month := month + 1;

        { The month in which the Julian_day falls equals the month found in the
          above 'while' loop.  The day of the month equals the difference
          between the Julian_day and the end of the previous month, hence the
          array index "month - 1". }
        day := Julian_date - end_of_month[ month - 1 ];

        { convert month value to a two character string, insert leading 0 if necessary }
        str( month : 2, month_str );
        month_str := '0' + delete_blanks( month_str );
        month_str := last_characters_in_string( month_str, 2 );

        { convert day value to a two character string, insert leading 0 if necessary }
        str( day : 2, day_str );
        day_str := '0' + delete_blanks( day_str );
        day_str := last_characters_in_string( day_str, 2 );

        Julian_to_numeric_date := month_str + day_str;

    end; { function Julian_to_numeric_date }


function date_string_from_numeric( numeric_date : numeric_date_string_type ) : line;
    {
      1. This function converts a date expressed as MMDD to the format
         "Month Day".  Note that the numeric date MMDD is passed as a character
         string even though MM and DD are integers.
      2. "Month" consists of the first three letters of the month and "Day"
```

```
        consists of the day-of-the-month expressed as an integer.
     3. An incorrect numeric date results in 'XXX 99' being returned.
     4. Example : numeric_date = 0329
                  date_string_from_numeric( '0329' ) returns MAR 29
   }
   const
       month_names : array[ 1..12 ] of string[3] =
                     ( 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN',
                       'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC' );
   var
       dummy, month_chars : line;
       month, condition_code : integer;
   begin

       { make sure the numeric date is in the correct format }
       dummy := delete_blanks( numeric_date );
       if   length( dummy ) = 3
       then dummy := '0' + dummy;

       { extract month value from character string }
       month_chars := dummy[ 1 ] + dummy[ 2 ];
       val( month_chars, month, condition_code );

       { In case the month value specifies a month in a subsequent year,
         'MOD' out the previous years to get a month value in the range
         1 to 12 }
       month := month MOD 12;
       if   month = 0
       then month := 12;

       if   ( ( condition_code = 0 ) and ( month in [ 1..12 ] ) )
       then date_string_from_numeric := month_names[ month ] + ' ' + dummy[ 3 ] + dummy[ 4 ]
       else date_string_from_numeric := 'XXX 99';

   end; { function date_string_from_numeric }


function date_string_from_Julian( Julian_date : integer ) : line;
   {
     This function converts a Julian date to a string of the format "Month Day".
     This function is the inverse of the function calculate_Julian_date.
     Example : date_string_from_numeric( 62 ) returns 'MAR 02'
   }
   begin

       date_string_from_Julian := date_string_from_numeric( Julian_to_numeric_date( Julian_date ) );

   end; { function date_string_from_Julian }


function calculate_Julian_date( date_as_a_string : line ) : integer;
```

```
{
  1. This function calculates the Julian date given a date in the form
     "Month Day" where "Month" consists of the first three letters of
     the month and "Day" consists of the day-of-the-month expressed as
     an integer.
  2. If the string submitted as a date does not result in a valid Julian
     date, the value -1 is returned.
  3. This function is the inverse of the function date_string_from_Julian.
  4. Example : calculate_Julian_date( 'Apr 14' ) returns 105

  Warning : The function does not check that the "Day" falls within the
            valid days of the given month; thus, for example, Apr 45 is
            accepted and the Julian day is calculated ( as if the date were
            entered as May 15 ).
}
const
    month_abbr = 'JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC';
    start_of_month : array[ 1..12 ] of integer =
                        ( 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 );
    numbers        : set of '0'..'9' = ['0'..'9'];
var
    dummy, squished_string           : line;
    date, day, month, OK_month, OK_day : integer;
    condition_code : integer;
begin

    squished_string := delete_blanks( date_as_a_string );

    { extract month field }
    dummy := copy( squished_string, 1, 3 );
    dummy := upper_case( dummy );
    OK_month := pos( dummy, month_abbr );

    { extract day field }
    dummy := copy( squished_string, 4, 2 );
    if   length( dummy ) = 1
    then dummy := '0' + dummy;
    if   ( ( dummy[ 1 ] in numbers ) and ( dummy[ 2 ] in numbers ) )
    then val( dummy, day, condition_code )
    else day := -1;

    if   ( ( OK_month > 0 ) and ( day > 0 ) )
    then
         begin
             month := 1 + round( ( OK_month - 1 ) / 3 );
             date := start_of_month[ month ] + day;
         end { if }
    else
         date := -1; { else }

    calculate_Julian_date := date;
    end; { function calculate_Julian_date }


function numeric_date_to_Julian( numeric_date : numeric_date_string_type ) : integer;
    {
      This function converts a date expressed as MMDD to a Julian date.
      Note that the numeric date is passed as a character string and the
      Julian date is returned as an integer.

      This function is the inverse of the function Julian_to_numeric_date.
    }
    begin

        numeric_date_to_Julian := calculate_Julian_date( date_string_from_numeric( numeric_date ) );

    end; { function numeric_date_to_Julian }


procedure CC_Get_DOS_Time( var Year, Month, Day, Hour, Min, Sec, Centi : Integer);
    {
      This procedure gets the system date and time from DOS.

      References : Borland International, "Turbo Pascal 3.0 Reference Manual"
                       Borland International, Scotts Valley, California, USA
                       1985  page 208

                   Peter Norton, "Inside the IBM-PC"
                       Robert J. Brady Co., Bowie, Maryland USA
                       1983  page 72

    }
    type RegPack = record
                      AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer;
                   end;

    var  Regs : RegPack;

    begin

        { system date }
        with Regs do
           begin
               AX := $2A00;
               MsDOS(Regs);

               Year  :=    CX ;
               Month := hi(DX);
               Day   := lo(DX);
           end;

        { system time }
        with Regs do
           begin
               AX := $2C00;
               MsDOS(Regs);

               Hour  := hi(CX);
               Min   := lo(CX);
               Sec   := hi(DX);
               Centi := lo(DX);
           end;

    end; { procedure CC_Get_DOS_Time }


procedure Julian_date_from_DOS( var this_season, this_Julian_date : integer );
    {
      This procedure returns the year and the Julian date of the date stored
      in the system, i.e. the DOS date.
    }
    var
        this_year, this_month, this_day, this_hour, this_minute, this_second, this_centi : integer;
        this_month_str, this_day_str : numeric_date_string_type;

    begin

        CC_Get_DOS_Time( this_year, this_month, this_day, this_hour, this_minute, this_second, this_centi );

        this_season := this_year;

        { convert month value to a two character string, insert leading 0 if necessary }
        str( this_month : 2, this_month_str );
        this_month_str := '0' + delete_blanks( this_month_str );
        this_month_str := last_characters_in_string( this_month_str, 2 );

        { convert day value to a two character string, insert leading 0 if necessary }
        str( this_day : 2, this_day_str );
        this_day_str := '0' + delete_blanks( this_day_str );
        this_day_str := last_characters_in_string( this_day_str, 2 );

        { construct a date in the format MMDD and convert it to a Julian date }
        this_Julian_date := numeric_date_to_Julian( this_month_str + this_day_str );

    end; { procedure Julian_date_from_DOS }


procedure fetch_Julian_date( date_prompt : line; numeric_date_string : numeric_date_string_type;
                             var Julian_date : integer; var response : line );
    {
     This procedure prompts the user to enter a date in the form Month Day, where
     Month is the 3 character abbreviation and Day is a number.  The prompt includes
     a default date embedded in the prompt string by the calling block and also
     given by the variable numeric_date_string.
```

```
  An immediate RETURN is a valid response which selects the default date.
  The string NOMORE is a valid response which causes an exit.
}
var
    upper_case_response : line;
begin

    repeat

         response := read_response( date_prompt );
         upper_case_response := upper_case( response );

         if   upper_case_response <> nomore
         then
              begin
                  if   response <> ''
                  then Julian_date := calculate_Julian_date( response )
                  else Julian_date := numeric_date_to_Julian( numeric_date_string );
                  if  Julian_date < 0
                  then writeln( beep + beep + beep + ' "', response, '" is an invalid date.' );
              end; { if response }

    until ( ( Julian_date > 0 ) or ( upper_case_response = nomore ) ); { repeat }

end; { procedure fetch_Julian_date }
```

```
INP-WTHR.PAS
------------

procedure input_weather;
    {
      This procedure prompts the user for input, performs some simple checks,
      and stores the weather data in the appropriate disk files.

      The procedure allows the user to enter weather data for any weather
      gathering site in any site order and for more than one site in a single
      data entry session.  Weather data may be entered in any date order and may
      be changed any number of times before being saved in the disk file.  There
      is no need for a separate verification procedure since the user merely
      scrolls through the newly entered data and re-enters values as required.

      Weather data are first read into internal arrays representing the entire
      year.  User input modifies these arrays.  Each array is written back to
      the disk after the user is finished entering data.  This method simplifies
      the processes of sorting the data by date and of storing the data in a
      disk file.  It also allows faster data entry since disk accesses are
      unnecessary during data entry.

      There are three primary arrays, the flags array and the temperature
      arrays.  The "flags" array marks ( flags ) the type of weather data for
      the indexed day.  "H" means historical or any data prior to the current
      year; "N" means normal ( averaged ) data; "A" means actual observed data;
      "P" means predicted data.  The max_temp and min_temp arrays contain the
      maximum and minimum temperatures respectively.
    }

    type
      { enter temperature as a string to allow some error checking }
      temperature_vector = array[ 1..366 ] of string[6];

    var
      max_temp, min_temp       : temperature_vector;
      dates                    : array[ 1..366 ] of numeric_date_string_type;
      flags                    : array[ 1..366 ] of string[3];
      response, prompt_string  : line;
      site_name                : disk_file_name;
      full_site_name           : full_site_name_type;
      site_exists              : boolean;
      site_directory_address   : integer;


    procedure fetch_weather_data( site_name : disk_file_name );
      {
        This procedure reads the temperature data for site "site_name" from
        the disk files into internal arrays.
      }
      var
        i         : integer;
        data_file : text;
        file_name : line;
    begin

        writeln;
        writeln( ' Fetching data. Please wait.' );

        { read flag file }
        file_name := PMS_root_directory + Year_path + site_name + '\WEATHER.FLG';
        assign( data_file, file_name );
        reset( data_file );
        for i := 1 to 366 do
            readln( data_file, dates[ i ], flags[ i ] );
        close( data_file );

        { read weather data file }
        file_name := PMS_root_directory + Year_path + site_name + '\WEATHER.DAT';
        assign( data_file, file_name );
        reset( data_file );
        for i := 1 to 366 do
            readln( data_file, dates[ i ], max_temp[ i ], min_temp[ i ] );
        close( data_file );

    end; { procedure fetch_weather_data }


procedure user_entered_weather_data( site_name : disk_file_name );
    {
      This procedure prompts the user to enter a date on which data are to
      be recorded and then prompts for the data.  The prompt includes a
      default value which is assumed to be the input if the user enters an
      immediate RETURN.
    }
    var
        date, Julian_date,
            earliest_date_entered          : integer;
        date_prompt, prompt                : line;
        data_file                          : text;
        site_path, file_name               : line;
        response, upper_case_response,
           weather_flag, flag_type         : line;
        temperature                        : real;
        condition_code                     : integer;
        set_of_weather_flags, actual_flag,
           subset_of_flags                 : set of 'A'..'Z';
        ch_flag                            : char;
        continue                           : boolean;

    begin

        set_of_weather_flags := [ 'H', 'N', 'A', 'P' ];
        actual_flag          := [ 'A' ];
```

```
site_path := PMS_root_directory + Year_path + site_name;

{ fetch the earliest date on which the user has entered weather data }
file_name :=  site_path + '\WEATHER.INF';
assign( data_file, file_name );
reset( data_file );
readln( data_file, earliest_date_entered );
close( data_file );

Julian_date := 1;  { start on January 01 }
date_prompt := ' Date ( ' + date_string_from_numeric( dates[ Julian_date ] ) + ' ) ? ';

repeat

     { prompt the user to enter a date; the default date is
       displayed in the format MMM DD, e.g. MAY 05 }
     fetch_Julian_date( date_prompt, dates[ Julian_date ], Julian_date, response );
     upper_case_response := upper_case( response );

     if   upper_case_response <> nomore
     then
          begin

              { earliest date on which user has entered data }
              if   earliest_date_entered > Julian_date
              then earliest_date_entered := Julian_date;

              { weather data flags }
              weather_flag := flags[ Julian_date ];
              ch_flag      := weather_flag[ 2 ];
              case ch_flag of
                  'H' : flag_type := 'HISTORICAL';
                  'N' : flag_type := 'NORMAL';
                  'A' : flag_type := 'ACTUAL';
                  'P' : flag_type := 'PREDICTED';
              end; { case ch_flag }

              if   Julian_date < todays_Julian_date
              then
                   subset_of_flags := set_of_weather_flags
              else
                   { cannot record Actual data on future days }
                   subset_of_flags := set_of_weather_flags - actual_flag;

              { fetch flag ( weather data type ) }
              repeat
                  prompt   := '    Type of data ( ' + flag_type + ' ) ? ';
                  response := delete_blanks( upper_case( read_response( prompt ) ) );
                  if   response <> ''
                  then
                       begin
```

```
             continue := ( response[ 1 ] in subset_of_flags );
             if   not continue
             then writeln( beep + beep + beep + '    ' + response +
                           ' cannot be interpreted as a valid type.' );
         end
    else
         continue := true;
until ( continue );

if   response <> ''
then weather_flag := ' ' + response[ 1 ];

{ fetch maximum temperature }
repeat

     condition_code := 0; { assume no error condition exists }

     prompt := '    Max. ( ' + max_temp[ Julian_date ] + ' ) ? ';
     response := read_response( prompt );
     response := delete_blanks( response );

     if   response <> ''
     then
          begin

              { convert input string to a real number }
              val( response, temperature, condition_code );

              if   condition_code = 0 { input string represents a real number }
              then { change the value in the internal array }
                   str( temperature : width_of_temperature_field : 1, max_temp[ Julian_date ] )
              else writeln( beep + beep + beep +
                            '    "', response, '" cannot be interpreted as a temperature.' );

          end;
     { else ( if response is null ), then do nothing;
       i.e. do not change the default value }

until ( condition_code = 0 );

{ fetch minimum temperature }
repeat

     condition_code := 0; { assume no error condition exists }

     prompt := '    Min. ( ' + min_temp[ Julian_date ] + ' ) ? ';
     response := read_response( prompt );
     response := delete_blanks( response );

     if   response <> ''
     then
          begin
```

```
                             { convert input string to a real number }
                             val( response, temperature, condition_code );

                             if   condition_code = 0 { input string represents a real number }
                             then { change the value in the internal array }
                                  str( temperature : width_of_temperature_field : 1, min_temp[ Julian_date ] )
                             else writeln( beep + beep + beep +
                                            '    "', response, '" cannot be interpreted as a temperature.' );

                         end;
                    { else ( if response is null ), then do nothing;
                      i.e. do not change the default value }

              until ( condition_code = 0 );

              flags[ Julian_date ] := weather_flag; { set flag indicating kind of weather data }

          end; { if response }

     Julian_date := Julian_date + 1;
     date_prompt := ' Date ( ' + date_string_from_numeric( dates[ Julian_date ] ) + ' ) ? ';

until ( upper_case_response = nomore ); { repeat }


writeln( ' Saving data.  Please wait.' );

{ store flags }
file_name := site_path + '\WEATHER.FLG';
assign( data_file, file_name );
rewrite( data_file );
for date := 1 to 366 do
    writeln( data_file, dates[ date ] : 4, flags[ date ] : 2 );
close( data_file );

{ store weather data }
file_name := site_path + '\WEATHER.DAT';
assign( data_file, file_name );
rewrite( data_file );
for date := 1 to 366 do
    writeln( data_file,    dates[ date ] : width_of_date_field,
                        max_temp[ date ] : width_of_temperature_field,
                        min_temp[ date ] : width_of_temperature_field );
close( data_file );

{ store earliest date on which user recorded weather data }
file_name := site_path + '\WEATHER.INF';
assign( data_file, file_name );
rewrite( data_file );
writeln( data_file, earliest_date_entered   ' );
close( data_file );
```

```
    end; { procedure user_entered_weather_data }


begin

    writeln;
    writeln( ' Task : Input Weather Data' );
    writeln;

    site_directory_address := 1;
    site_name := master_directory[ site_directory_address ].site_name;

    { use this procedure to search for the first site which gathers weather
      data ( in case the first site does not ) and to initialize the other
      site parameters }
    valid_weather_site( site_name, full_site_name, site_exists, site_directory_address );

    {
      This repeat loop allows the user to enter data for a site more than
      once in the same "input weather data" session because the loop does
      not stop once it has returned to the beginning.  The only stop
      condition is the user entered "NOMORE".
    }
    repeat

        fetch_site_name( ' Weather', response, site_name, full_site_name, site_exists ,site_directory_address );

        if   response <> nomore
        then
             begin

                 if   found_in_directory( site_name, site_directory_address, weather )
                 then
                      begin
                          fetch_weather_data( site_name );
                          user_entered_weather_data( site_name );

                          { increment to next site in master_directory }
                          increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );
                      end
                 else
                      writeln( beep + ' Requested site does not gather weather data.' );

                 { make sure the next site gathers weather data; if it does not,
                   then increment to the next site which does }
                 valid_weather_site( site_name, full_site_name, site_exists, site_directory_address );

             end; { if response }

    until ( response = nomore );
```

```
end; { procedure input_weather }
```

```
INP-TRAP.PAS
------------

procedure input_trap_catch;
    {
      This procedure prompts the user for input, performs some simple checks,
      and stores the insect trap count data in the appropriate disk file.  Data
      are stored in descending date order.

      The procedure allows the user to enter trap data for any insect trap
      count gathering site in any site order and for more than one site in a
      single data entry session.  Trap data may be entered in any date order
      and may be changed any number of times before being saved in the disk
      file.  There is no need for a separate verification procedure since the
      user merely scrolls through the newly entered data and re-enters values
      as required.

      Trap data are first read into an internal array representing the entire
      year.  User input modifies this array.  The array is written back to the
      disk after the user is finished entering data.  This method simplifies
      the processes of sorting the data by date and of storing the data in a
      disk file.  It also allows faster data entry since disk accesses are
      unnecessary during data entry.

      Unlike weather data files, trap count files do not necessarily contain
      data for every day of the year.  Only recorded trap counts are stored in
      the disk file. If a date is unrepresented, i.e. the data do not appear in
      the file, then no trap data were gathered on that date. Note : Zero
      counts are data and are recorded as 0's.  Zero is not null.
    }

    var
        response, prompt_string,
           upper_case_pest_response, pest_response,
           pest_prompt, pest_name                 : line;
        site_name, pest_file_name                 : disk_file_name;
        full_site_name                            : full_site_name_type;
        site_directory_address, max_traps         : integer;
        site_exists, valid_pest, OK_trap_site     : boolean;
        pest_type, old_pest_type, dummy_pest_type : kinds_of_sites;
        first_record_in_trap_file                 : string[ 255 ];
        dates                                     : array[ 1..366 ] of numeric_date_string_type;
        dates_str                                 : numeric_date_string_type;
        trap_records                              : array[ 1..366 ] of line;
        trap_records_str                          : line;

        { simple statistics for each trap count record }
        stats                                     : array[ 1..366 ] of string[ width_of_stats_field ];
        stats_str                                 : string[ width_of_stats_field ];

    procedure next_trapped_pest( var pest_prompt, pest_name : line;
                              var pest_file_name : disk_file_name; var pest_type : kinds_of_sites);
    {
      This procedure increments from the current pest ( passed as the
      parameter "pest_type" ) to the next pest on which the site "site_name"
      gathers trap data.  The pest type, disk file name in which the trap
      counts are stored, and pest name are returned.

      This procedure assumes the site gathers trap data for at least
      one pest.

      Note : The site parameters "site_name" and "site_directory_address"
             are global to this procedure and are not explicitly passed to
             it.
    }
    var
        OK_pest      : boolean;
        trapped_pest : kinds_of_sites;
    begin

        OK_pest := false;

        trapped_pest := pest_type;

        repeat
             trapped_pest := succ( trapped_pest );

             { If at the end of the pest list, go back to the beginning. }
             if   trapped_pest = last_pest
             then trapped_pest := succ( first_pest );

             OK_pest := found_in_directory( site_name, site_directory_address, trapped_pest );
        until ( OK_pest );

        pest_type      := trapped_pest;
        pest_file_name := pest_file_name_vector[ pest_type ];
        pest_name      := pest_name_vector[ pest_type ];
        pest_prompt    := ' Pest ( ' + pest_name + ' ) ? ';

    end; { procedure next_trapped_pest }


procedure fetch_trap_data( site_name, pest_name : disk_file_name );
    {
      This procedure reads the trap count data for pest "pest_name" at site
      "site_name" from the appropriate disk file.
    }
    var
        i         : integer;
        data_file : text;
        file_name : line;

    begin
```

```
        file_name := PMS_root_directory + Year_path + site_name + '\' + pest_name + '.TRP';

        writeln;
        writeln( ' Fetching data. Please wait.' );

        { initialize elements of the trap count arrays to null strings }
        for i := 1 to 366 do
            begin
                stats[ i ]        := '';
                trap_records[ i ] := '';
            end; { for i }

        assign( data_file, file_name );
        reset( data_file );

        {
          The first record of the trap count disk file contains dummy data.
          The number of trap count fields tells the program how many traps
          are used at this site; this value is stored in the variable
          "max_traps".  This allows the program to prompt for the correct
          number of trap counts.
          Note : This also allows the user to change the number of traps by
                 merely changing the first record of the disk data file;
                 the user does not have to change the program.
        }
        readln( data_file, first_record_in_trap_file );
        max_traps := round( ( length( first_record_in_trap_file ) - ( width_of_date_field + width_of_stats_field ) ) /
                              width_of_trap_field );

        while ( not eof( data_file ) ) do
            begin
                readln( data_file, dates_str, stats_str, trap_records_str );
                i                 := numeric_date_to_Julian( dates_str );
                dates[ i ]        := dates_str;
                stats[ i ]        := stats_str;
                trap_records[ i ] := trap_records_str;
            end; { while }

        close( data_file );

    end; { procedure fetch_trap_data }


procedure user_entered_trap_data( site_name, pest_name ; disk_file_name );
    {
      This procedure prompts the user to enter a date on which data are to
      be recorded and then prompts for the data.  The prompt includes a
      default value which is assumed to be the input if the user enters an
      immediate RETURN.

      A data record cannot be removed from the file.  An erroneous date can
```

```
  be changed only by manually editing the trap count disk file.
}
const
    zero_trap_record = '      0';
var
    i, trap, date, Julian_date, condition_code, count : integer;
    date_prompt, prompt, dummy_record, dummy,
            old_trap_record,
            date_response, upper_case_response,
            trap_response, massaged_response,
            several_blanks                          : line;
    file_name                                       : line;
    data_file                                       : text;
    trap_count                                      : array[ 1..grand_maximum_traps ] of string[ width_of_trap_field ];
    missing, positive_count                         : boolean;

begin

    several_blanks := '';
    for i := 1 to width_of_trap_field do several_blanks := ' ' + several_blanks;

    Julian_date := 1;

    repeat

        date_prompt := ' Date ( ' + date_string_from_Julian( Julian_date ) + ' ) ? ';
        fetch_Julian_date( date_prompt, Julian_to_numeric_date( Julian_date ), Julian_date, date_response );
        upper_case_response := upper_case( date_response );

        if   upper_case_response <> nomore
        then
             begin

                 { extract trap record }
                 old_trap_record := trap_records[ Julian_date ];
                 dummy_record    := old_trap_record;
                 if   dummy_record = ''
                 then
                      { default trap count for a new record is 0 }
                      for i := 1 to max_traps do
                          dummy_record := dummy_record + zero_trap_record;

                 { initialize trap_count vector by extracting individual fields of the trap record }
                 for trap := 1 to max_traps do
                     begin
                         i := width_of_trap_field * ( trap - 1 ) + 1;
                         trap_count[ trap ] := copy( dummy_record, i, width_of_trap_field );
                     end; { for trap }

                 { Use "dummy_record" to build a trap file data record from the user's inputs. }
                 dummy_record := '';
```

```
for trap := 1 to max_traps do
    begin

        repeat

             condition_code := 0;     { assume nothing wrong yet }
             positive_count := true; { assume trap count entry is non-negative }

             str( trap : 2, dummy ); { convert integer to string }
             prompt := '     Trap ' + dummy + ' ( ' + trap_count[ trap ] + ' ) ? ';
             trap_response := read_response( prompt );
             trap_response := delete_blanks( trap_response );

             { Missing data are flagged by an "M" or "m" ( for "MISSING" ). }
             missing := ( pos( 'M', trap_response ) > 0 ) or ( pos( 'm', trap_response ) > 0 );

             if   trap_response <> ''
             then

                  if   not missing
                  then
                       begin
                           val( trap_response, count, condition_code );
                           if   condition_code = 0
                           then
                                if   count >= 0  { check for non-negative trap count }
                                then
                                     { Pad the trap count entry with leading blanks to ensure
                                       that it will fit in the trap-count-field properly. }
                                     massaged_response := several_blanks + trap_response
                                else
                                     begin
                                         positive_count := false;
                                         writeln( beep + beep + beep +
                                                  '     Trap count must be non-negative.' );
                                     end
                           else
                                writeln( beep + beep + beep +
                                         '    "', trap_response,
                                              '" cannot be interpreted as an integer.' );
                       end { if not missing }

                  else { if not missing }
                       { Pad the word "MISSING" with leading blanks to ensure
                         that it will fit in the trap-count-field properly. }
                       massaged_response := several_blanks + 'MISSING'

             else { if trap_response }
                  { An immediate RETURN causes the program to assume
                    the default value as the user entered one. }
                  massaged_response := trap_count[ trap ];
```

```
                                until ( ( condition_code = 0 ) and positive_count );

                                trap_count[ trap ] := last_characters_in_string( massaged_response, width_of_trap_field );
                                dummy_record := dummy_record + trap_count[ trap ];

                            end; { for trap }

                        { Once the new trap record "dummy_record" has been built successfully,
                          the old trap record may be replaced with the new one. }
                        trap_records[ Julian_date ] := dummy_record;

                        if   ( dummy_record <> old_trap_record ) { may have to recalculate statistics for }
                        then stats[ Julian_date ] := ' ';        { an old record which has been changed   }

                        Julian_date := Julian_date + 1;

                   end; { if upper_case_response }

         until ( upper_case_response = nomore ); { repeat }


         writeln( ' Saving data.  Please wait.' );

         file_name := PMS_root_directory + Year_path + site_name + '\' + pest_name + '.TRP';
         assign( data_file, file_name );
         rewrite( data_file );

         {
           This file must end at the end of a complete trap record, not on a blank line.
           Therefore, a "writeln" appears before each "write" of a trap record.
         }
         write( data_file, first_record_in_trap_file );
         for date := 366 downto 1 do               { store data in reverse date order }
             if   trap_records[ date ] <> ''
             then
                  begin
                      writeln( data_file );
                      write( data_file, Julian_to_numeric_date( date ) : width_of_date_field,
                                        stats[ date ] : width_of_stats_field, trap_records[ date ] );
                  end; { if trap_records }

        close( data_file );

    end; { procedure user_entered_trap_data }


begin

    writeln;
    writeln( ' Task : Input Trap Catch Data' );
    writeln;
```

```
site_directory_address := 1;
site_name := master_directory[ site_directory_address ].site_name;

repeat

     { make sure the site gathers pest trap data }
     next_trap_site( site_name, site_directory_address );

     repeat
          fetch_site_name( ' Pest trapping', response, site_name,full_site_name, site_exists,site_directory_address );
          if   response <> nomore
          then
               begin
                   OK_trap_site := valid_trap_site( site_name, site_directory_address );
                   if   not OK_trap_site
                   then
                        begin
                            writeln( beep + ' Requested site does not trap any pests.' );
                            next_trap_site( site_name, site_directory_address );
                        end; { if not OK_trap_site }
               end; { if response }
     until ( OK_trap_site or ( response = nomore ) );

     if   response <> nomore
     then
          begin

               pest_type := first_pest;
               next_trapped_pest( pest_prompt, pest_name, pest_file_name, pest_type );

               { The following repeat loop cycles through the list of pests on which data are
                 gathered at the current site.  After the last pest is reached the loop starts
                 at the beginning of the list again.  The loop repeats until the user enters
                 "NOMORE". }
               repeat

                    old_pest_type := pest_type;

                    { The following repeat loop prompts the user to enter a pest name,
                      checks that it is one which the program recognizes, and, if necessary,
                      reprompts the user until a recognized pest or "NOMORE" is entered. }

                    repeat

                         dummy_pest_type := old_pest_type;

                         { user entered pest name }
                         pest_response := read_response( pest_prompt );
                         upper_case_pest_response := upper_case( delete_blanks( pest_response ) );
                         { immediate RETURN assumes user is selecting dummy_pest_type }
                         if   upper_case_pest_response = ''
```

```
      then upper_case_pest_response := delete_blanks( pest_name_vector[ dummy_pest_type ] );

      if   upper_case_pest_response <> nomore
      then
           begin

               { make sure user entered pest is recognized by the program }
               dummy_pest_type := succ( first_pest );
               repeat
                    valid_pest := ( upper_case_pest_response =
                                    delete_blanks(pest_name_vector[ dummy_pest_type ]) ) or

                                  ( upper_case_pest_response =
                                    pest_name_vector_abbr[ dummy_pest_type ] );

                    dummy_pest_type  := succ( dummy_pest_type );
               until ( ( dummy_pest_type = last_pest ) or valid_pest );

               { "dummy_pest_type" is incremented once too often;
                 hence, the preceeding pest type must be selected. }
               dummy_pest_type := pred( dummy_pest_type );

               if   not valid_pest
               then writeln( beep + ' Pest "', pest_response, '" is not a trapped pest.' );

          end; { if upper_case_pest_response }

until ( valid_pest or ( upper_case_pest_response = nomore ) );

if   upper_case_pest_response <> nomore
then
     begin

         { Reset pest name, pest file name, etc. to the user selected pest. }
         pest_type := dummy_pest_type;
         pest_name := pest_name_vector[ pest_type ];
         pest_file_name := pest_file_name_vector[ pest_type ];
         pest_prompt := ' Pest ( ' + pest_name + ' ) ? ';

         if   found_in_directory( site_name, site_directory_address, pest_type )
         then
              begin
                  fetch_trap_data( site_name, pest_file_name );
                  user_entered_trap_data( site_name, pest_file_name );
              end
         else
              writeln( beep + ' Requested site does not trap "', pest_name, '".' );

         next_trapped_pest( pest_prompt, pest_name, pest_file_name, pest_type );

     end; { if upper_case_pest_response }
```

```
                   until ( upper_case_pest_response = nomore );
                   increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );
              end; { if response }
    until ( response = nomore );
end; { procedure input_trap_catch }
```

CALCULAT.PAS

```
procedure calculate_lifestage_development;
    {
      This procedure calculates the development to date of all active lifestages
      and forecasts development for days on which forecast weather data have
      been entered.  An active lifestage is defined to be one which has started
      development and is < 110% developed.
    }

    const
        end_active_development = 1.10; { end an "active" lifestage at 110% development }
        maximum_lifestages = 10;
    type
        temperature_vector = array[ 1..366 ] of real;

        { The developmental rate function is expressed as a vector.  The
          vector's index is the temperature and the vector element is the
          development rate.  For now, assume the temperature range [ 0, 40 ] is
          sufficient to cover the entire developmental range. }
        rate_function      = array[ 0..40 ] of real;

    var
        dates                     : array[ 1..366 ] of string[ width_of_date_field ];
        max_temp, min_temp        : temperature_vector;
        weather_flag              : array[ 1..366 ] of string[ 3 ];
        i, temp, d100, today,
         lifestage, n_lifestages : integer;

        { *.RAT file values.  This file stores the rate of development for each
          lifestage at each temperature. }
        rate_matrix               : array[ 1.. maximum_lifestages ] of rate_function;

        { *.MDL file values.  This file contains the information describing the
          development of the pest as modelled by the program. }
        active_flag               : array[ 1.. maximum_lifestages ] of char;
        generation_number         : array[ 1.. maximum_lifestages ] of string[ 2 ];
        lifestage_name            : array[ 1.. maximum_lifestages ] of string[ 30 ];
        development_last_actual, development_last_predicted : array[ 1.. maximum_lifestages ] of real;
        { dates on which certain events occur }
        date_000_modelled, date_000_observed,               { date on which   0% development occurs ( start date ) }
            date_100_modelled, date_100_observed,           { date on which 100% development occurs ( end date )   }
            date_end_active_modelled, date_end_active_observed,  { end of the active lifestage }
            date_last_actual, date_last_predicted           { most recent date processed or forecast }
                                  : array[ 1.. maximum_lifestages ] of integer;

        dummy, site_path, file_name, init_MDL_file : line;
        RAT_file, MDL_file, INF_file               : text;
        site_name                                  : disk_file_name;
        full_site_name                             : full_site_name_type;
        site_exists                                : boolean;
    site_directory_address                : integer;
    pest_type                             : kinds_of_sites;
    earliest_date_entered                 : integer;

    debugstage : integer;

procedure fetch_weather_data_real( site_name : disk_file_name );
    {
      This procedure reads the weather data from the appropriate disk file
      and stores them in internal arrays.
    }
    var
        i         : integer;
        data_file : text;
        file_name : line;
    begin

        file_name := PMS_root_directory + Year_path + site_name + '\WEATHER.FLG';
        assign( data_file, file_name );
        reset( data_file );
        for i := 1 to 366 do
            readln( data_file, dates[ i ], weather_flag[ i ] );
        close( data_file );

        file_name := PMS_root_directory + Year_path + site_name + '\WEATHER.DAT';
        assign( data_file, file_name );
        reset( data_file );
        for i := 1 to 366 do
            readln( data_file, dates[ i ], max_temp[ i ], min_temp[ i ] );
        close( data_file );

    end; { procedure fetch_weather_data_real }


function daily_development( Julian_day : integer; rate_vector : rate_function ) : real;
    {
      This function calculates the development of a lifestage over one day
      by integrating the developmental rate function over time ( 24 hours ).
      The function performs a numerical integration using Simpson's rule.

      Global vector variables "max_temp" and "min_temp" are used.

      Assumptions :
          1. Temperature minimum occurs at 0500.
          2. Temperature maximum occurs at 1400.
          3. Sine curves connect the maxima and minima.  Therefore, there are
             three separate sine curves which describe the temperature curve over
             the entire day.
          4. Linear interpolation is used to calculate developmental rates
             for non-integer temperatures.  Some sort of interpolation is
             required since the rate function is expressed as a vector
             indexed by integer values of temperature.
```

```
  Reference for Simpson's Rule
      R.C. Weast, editor
      Handbook of Tables for Mathematics
      Chemical Rubber Company
      Cleveland, Ohio, USA 44128
      4th edition, 1970, page 14

}
const
    twopi        = 6.28318;
    hours_in_day = 24;
    delta_time   = 2.0; { hours }

    { time is measured in hours with respect to midnight ( 0 hrs. ) this morning }
    time_of_max_yesterday = -10.0;
    time_of_min_today     =   5.0; { time at which minimum temperature occurs }
    time_of_max_today     =  14.0; { time at which maximum temperature occurs }
    time_of_min_tomorrow  =  29.0;

var
    max_temp_yesterday, min_temp_today,
         max_temp_today, min_temp_tomorrow    : real;

    { Three sine curves are used to describe the temperature curve over one day. }
    amplitude, mean, phase, omega             : array[ 1..3 ] of real;

    sum, time_t, temperature, development_rate : real;
    Simpson_weight                            : real;
    lower, upper                              : integer;


function piecewise_sine_curve( i : integer; time_t : real ) : real;
    {
      This function returns the value of the sine curve for a given
      time t.  The function selects the correct sine curve from the
      three curves used to describe the entire day's temperature curve.
    }
    begin

        piecewise_sine_curve := mean[ i ] + amplitude[ i ] * sin( omega[ i ] * ( time_t - phase[ i ] ) );

    end; { function piecewise_sine_curve }


function interpolate_development_rate( temperature : real ) : real;
   {
     This function uses linear interpolation to calculate development
     rates from the development rate vector for non-integer
     temperatures.  The rate vector is passed to this function as
     a global variable.
```

```
      Note : The function returns a value of 0.0 for temperatures
             outside of the rate vector's index range.
    }
    var
        lower, upper : integer;
        rate         : real;
    begin

        lower := trunc( temperature );
        upper := trunc( temperature + 1.0 );

        { The expression "( rate_vector[ upper ] - rate_vector[ lower ] )" should be divided by the
          expression "( upper - lower )" in a completely general linear interpolation
          formula.  For this application ( upper - lower ) always equals 1.0, therefore,
          division by "( upper - lower )" is unnecessary.
        }
        if   ( ( 0.0 < temperature ) and ( temperature < 40.0 ) )
        then rate := rate_vector[ lower ] + ( temperature - lower ) * ( rate_vector[ upper ] - rate_vector[ lower ] )
        else rate := 0.0;

        interpolate_development_rate := rate;

    end; { function interpolate_development_rate }


begin

    if   Julian_day > 1
    then max_temp_yesterday := max_temp[ Julian_day - 1 ]
    else
         { if   today is 01 January
           then assume yesterday's temperature is the same as today's }
         max_temp_yesterday := max_temp[ Julian_day ];

    min_temp_today := min_temp[ Julian_day ];
    max_temp_today := max_temp[ Julian_day ];

    if   Julian_day < 366
    then min_temp_tomorrow := min_temp[ Julian_day + 1 ]
    else
         { if   today is 31 December
           then assume tomorrow's temperature is the same as today's }
         min_temp_tomorrow := min_temp[ Julian_day ];


    { construct the parameters for the three separate sine curves }

    mean[ 1 ]      := ( min_temp_today + max_temp_yesterday ) * 0.5;
    amplitude[ 1 ] := ( min_temp_today - max_temp_yesterday ) * 0.5;
    phase[ 1 ]     := ( time_of_min_today + time_of_max_yesterday ) * 0.5;
    omega[ 1 ]     := twopi 7 ( time_of_min_today - time_of_max_yesterday );
```

```
mean[ 2 ]       := ( max_temp_today + min_temp_today ) * 0.5;
amplitude[ 2 ] := ( max_temp_today - min_temp_today ) * 0.5;
phase[ 2 ]      := ( time_of_max_today + time_of_min_today ) * 0.5;
omega[ 2 ]      := twopi 7 ( time_of_max_today - time_of_min_today );

mean[ 3 ]       := ( min_temp_tomorrow + max_temp_today ) * 0.5;
amplitude[ 3 ] := ( min_temp_tomorrow - max_temp_today ) * 0.5;
phase[ 3 ]      := ( time_of_min_tomorrow + time_of_max_today ) * 0.5;
omega[ 3 ]      := twopi 7 ( time_of_min_tomorrow - time_of_max_today );


{ Simpson's rule numeric integration }

sum := 0.0;

{ function value at the left endpoint of the domain, 0 Hours }
temperature      := piecewise_sine_curve( 1 , 0.0 );
development_rate := interpolate_development_rate( temperature );
sum              := sum + development_rate * delta_time;

{ function value at the right endpoint of the domain, 24 hours }
temperature      := piecewise_sine_curve( 3 , 24.0 );
development_rate := interpolate_development_rate( temperature );
sum              := sum + development_rate * delta_time;

time_t := delta_time;
Simpson_weight := 4.0;

while  ( time_t < hours_in_day ) do
    begin

        { select the sine curve which describes the temperature curve at the current time }
        if   ( time_t < time_of_min_today )
        then i := 1;
        if   ( ( time_of_min_today < time_t ) and ( time_t < time_of_max_today ) )
        then i := 2;
        if   ( time_of_max_today < time_t )
        then i := 3;

        {
          calculate the development rate at the current time;
            a. determine the temperature at the current time
            b. determine the development rate at the calculated temperature
        }
        temperature := piecewise_sine_curve( i, time_t );
        development_rate := interpolate_development_rate( temperature );

        sum    := sum + Simpson_weight * development_rate * delta_time;
        time_t := time_t + delta_time;

        if   round( Simpson_weight ) = 4
        then Simpson_weight := 2.0
```

```
                else Simpson_weight := 4.0;

            end; { while }

        {
          The development time units are days while the integration time
          units are hours.  This means the accumulated sum must be divided
          by the number of hours in the day in order to convert from hours
          to days.
        }
        sum := sum / ( 3.0 * hours_in_day );

        daily_development := sum;

    end; { function daily_development }


procedure active_lifestage_development( start_development_date : integer; rate_vector : rate_function;
                                        var day_development100, day_end_active_development : integer;
                                        var day_last_actual : integer; var development_last_actual : real;
                                        var day_last_predicted : integer; var development_last_predicted : real );
    {
      This procedure calculates the development to date of a lifestage and
      forecasts development for future days on which forecast weather data
      have been entered.  Calculations stop when development reaches 100%
      and the end-active-lifestage percentage.
    }

    const
        development100          = 1.00; { 100% development }

    var
        day, stop_calc_day, next_begin_day : integer;
        development                                  : real;
        flag_100, flag_end_active_development        : boolean;

    begin

        flag_100                    := false; { assume 100% development not yet reached }
        flag_end_active_development := false; { assume end of active development not yet reached }

        development := 0.0;
        day         := start_development_date;

        {
          Calculate actual development units for days on which actual
          weather data were recorded - only for active lifestages.  The while
          loop stops at the first day which is not flagged 'A' even if that
          day is distant in the past.  Thus, actual development is
          calculated on an unbroken series of days on which actual weather
          data have been entered.
        }
```

```
while  ( ( weather_flag[ day ] = ' A' ) and ( not flag_end_active_development ) ) do
    begin

        development := development + daily_development( day, rate_vector );

        if   not flag_100
        then
             { if   development passed 100% today,
               then flag this event and save this date }
             if   development > development100
             then
                  begin
                      flag_100            := true;
                      day_development100 := day;
                  end; { if development }

        if   not flag_end_active_development
        then
             { if   development passed end_active_development percentage today,
               then flag this event and save this date }
             if   development > end_active_development
             then
                  begin
                      flag_end_active_development := true;
                      day_end_active_development  := day;
                  end; { if flag_100 }

        day := day + 1;

    end; { while weather_flag }

development_last_actual := development; { development on the most recent date of actual weather data }
day_last_actual         := day - 1;     { most recent date of actual weather data }


{
  Calculate development units for days on which forecast weather
  data were recorded - only for active lifestages.  The while loop
  stops at the first day which is not flagged 'P'.  Thus forecast
  development covers an unbroken series of days on which forecast
  weather data have been entered.
}
while  ( ( weather_flag[ day ] = ' P' ) and ( not flag_end_active_development ) ) do
    begin

        development := development + daily_development( day, rate_vector );

        if   not flag_100
        then
             { if   development passed 100% today,
               then flag this event and save this date.
               Note : this is a predicted date. }
```

```
                     if   development > development100
                     then
                          begin
                              flag_100            := true;
                              day_development100 := day;
                          end; { if development }

                if   not flag_end_active_development
                then
                     { if   development passed end_active_development percentage today,
                       then flag this event and save this date
                       Note : this is a predicted date. }
                     if   development > end_active_development
                     then
                          begin
                              flag_end_active_development := true;
                              day_end_active_development  := day;
                          end; { if flag_100 }

                day := day + 1;

            end; { while weather_flag }

        development_last_predicted := development; { development on the most future date of forecast weather data }
        day_last_predicted         := day - 1;     { most future date of forecast weather data }

    end; { procedure active_lifestage_development }


procedure read_MDL_file( file_name : line; n_lifestages : integer );
    {
      A procedure to read an MDL file.  The lifestage development variables
      are passed as global variables.
    }
    var
        lifestage : integer;
    begin

        assign( MDL_file, file_name );
        reset( MDL_file );

        for lifestage := 1 to n_lifestages do
            readln( MDL_file,
                   active_flag[ lifestage ], generation_number[ lifestage ],
                   lifestage_name[ lifestage ],
                   date_last_actual[ lifestage ], development_last_actual[ lifestage ],
                   date_last_predicted[ lifestage ], development_last_predicted[ lifestage ],
                   date_000_modelled[ lifestage ], date_000_observed[ lifestage ],
                   date_100_modelled[ lifestage ], date_100_observed[ lifestage ],
                   date_end_active_modelled[ lifestage ], date_end_active_observed[ lifestage ] );

        close( MDL_file );
    end; { procedure read_MDL_file }

begin

    today := todays_Julian_date;

    site_directory_address := 1;
    site_name := master_directory[ site_directory_address ].site_name;
    full_site_name := master_directory[ site_directory_address ].full_name;

    { search through entire list of sites }
    repeat

        { perform weather model calculations only at weather sites }
        if   master_directory[ site_directory_address ].trapped[ weather ]
        then
            begin

                { read weather files }
                fetch_weather_data_real( site_name );

                pest_type := succ( first_pest );

                repeat

                    { perform weather model calculations only on modelled pests }
                    if   master_directory[ site_directory_address ].modelled[ pest_type ]
                    then
                        begin

                            site_path := PMS_root_directory + Year_path + site_name + '\';

                            { read WEATHER.INF file to get earliest date on which user has entered weather data }
                            file_name := site_path + 'WEATHER.INF';
                            assign( INF_file, file_name );
                            reset( INF_file );
                            readln( INF_file, earliest_date_entered );
                            close( INF_file );

                            file_name := site_path + pest_file_name_vector[ pest_type ];

                            { read *.RAT file to get rate matrix }
                            assign( RAT_file, file_name + '.RAT' );
                            reset( RAT_file );

                            readln( RAT_file, dummy ); { first line used to determine number of lifestages }
                            { Note : The rate index field is the temperature field. }
                            n_lifestages := round( ( length( dummy ) - ( width_of_rate_index_field ) ) /
                                                                       width_of_rate_field );
```

```
for temp := 0 to 40 do
    begin
        read( RAT_file, i ); { read temperature field, but ignore it }
        for lifestage := 1 to n_lifestages do
            read( RAT_file, rate_matrix[ lifestage, temp ] );
        readln( RAT_file );
    end; { for temp }

close( RAT_file );

{ read *.MDL file to get lifestage development }
read_MDL_file( file_name + '.MDL', n_lifestages );


{
  This block compares the ealiest date entered against the most recent date
  on which a lifestage was deactivated.  If the earliest date entered is
  prior to the deactivation date, then the lifestage development
  calculations must be redone because the lifestage completion dates may
  change given different weather data.  For programming ease, the entire
  set of lifestage development variables is reinitialized.
}
{ search for most recently deactivated lifestage }
lifestage := n_lifestages + 1;
repeat
    lifestage := lifestage - 1;
until ( ( active_flag[ lifestage ] = 'P' ) or ( lifestage = 1 ) );

if   active_flag[ lifestage ] = 'P'
then
     if   earliest_date_entered < date_100_modelled[ lifestage ]
     then
          begin
              {
                initialize lifestage development to start-of-season
                values by reading the INITSITE *.MDL file
              }
              init_MDL_file := PMS_root_directory + Year_path + 'INITSITE\' +
                               pest_file_name_vector[ pest_type ] + '.MDL';
              read_MDL_file( init_MDL_file, n_lifestages );

          end; { if earliest_date_entered < date_100_modelled[ lifestage ] }
{ if active_flag[ lifestage ] = 'P' }


{ active lifestage calculations }

for lifestage := 1 to n_lifestages do
    begin

        if   ( active_flag[ lifestage ] = 'H' )
        then
```

```
            if   date_000_modelled[ lifestage ] <= today
            then { wake up a hibernating lifestage }
                 active_flag[ lifestage ] := 'A';

        if   ( active_flag[ lifestage ] = 'A' )
        then
            begin

                active_lifestage_development( date_000_modelled[ lifestage ],
                     rate_matrix[ lifestage ], date_100_modelled[ lifestage ],
                     date_end_active_modelled[ lifestage],
                     date_last_actual[ lifestage ], development_last_actual[ lifestage ],
                     date_last_predicted[ lifestage ],
                     development_last_predicted[ lifestage ] );

               d100 := date_100_modelled[ lifestage ];

               if   d100 <= today
               then
                    {
                      If   the current lifestage has completed 100% development by today,
                      then activate the next lifestage and store the start date which is
                      the same as today, the 100% development date of the current lifestage.
                    }
                    if   lifestage < n_lifestages
                    then { Do not perform this if the current lifestage is the last one. }
                         begin
                             active_flag[ lifestage + 1 ] := 'A'; { activate next stage }
                             date_000_modelled[ lifestage + 1 ]  := d100;
                         end; { if lifestage }

               { If the current lifestage has developed beyond the end_active
                 development percentage, then deactivate the lifestage. }
               if   date_end_active_modelled[ lifestage ] <= today
               then active_flag[ lifestage ] := 'P';

           end; { if active_flag }

    end; { for lifestage }

{ write new *.MDL file }
assign( MDL_file, file_name + '.MDL' );
rewrite( MDL_file );

for lifestage := 1 to n_lifestages - 1 do
    writeln( MDL_file,
           active_flag[ lifestage ]               : 1,
           generation_number[ lifestage ]         : 2,
           lifestage_name[ lifestage ]            : 30,
           date_last_actual[ lifestage ]          : 4,
           development_last_actual[ lifestage ]   : 8 : 4,
           date_last_predicted[ lifestage ]       : 4,
                                       development_last_predicted[ lifestage ] : 8 : 4,
                                       date_000_modelled[ lifestage ]           : 4,
                                       date_000_observed[ lifestage ]           : 4,
                                       date_100_modelled[ lifestage ]           : 4,
                                       date_100_observed[ lifestage ]           : 4,
                                       date_end_active_modelled[ lifestage ]    : 4,
                                       date_end_active_observed[ lifestage ]    : 4
                                       );

                            { end of line of the last record == end of file, hence "write" and not "writeln" }
                            lifestage := n_lifestages;
                            write( MDL_file,
                                   active_flag[ lifestage ]                : 1,
                                   generation_number[ lifestage ]          : 2,
                                   lifestage_name[ lifestage ]             : 30,
                                   date_last_actual[ lifestage ]           : 4,
                                   development_last_actual[ lifestage ]    : 8 : 4,
                                   date_last_predicted[ lifestage ]        : 4,
                                   development_last_predicted[ lifestage ] : 8 : 4,
                                   date_000_modelled[ lifestage ]          : 4,
                                   date_000_observed[ lifestage ]          : 4,
                                   date_100_modelled[ lifestage ]          : 4,
                                   date_100_observed[ lifestage ]          : 4,
                                   date_end_active_modelled[ lifestage ]   : 4,
                                   date_end_active_observed[ lifestage ]   : 4
                                   );

                            close( MDL_file );

                        end; { if master_directory.. modelled }

                   pest_type := succ( pest_type );

              until ( pest_type = last_pest );

          end; { if master_directory.. trapped[ weather ] }

     increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

until ( site_directory_address = 1 ); { back to the beginning }

{
  reset WEATHER.INF files

  After all the lifestage development calculations are finished, the earliest
  date on which the user entered weather data is meaningless and must be reset
  so that a subsequent call to "input Weather data" will give a new "earliest date"
  value.
}
site_directory_address := 1;
site_name := master_directory[ site_directory_address ].site_name;
repeat
```

```
              if   master_directory[ site_directory_address ].trapped[ weather ]
              then
                   begin
                       file_name := PMS_root_directory + Year_path + site_name + '\WEATHER.INF';
                       assign( INF_file, file_name );
                       rewrite( INF_file );
                       writeln( INF_file, ' 366' ); { any date in the year is not later than Julian date 366 }
                       close( INF_file );
                   end; { if master_directory[ site_directory_address ].trapped[ weather ] }

              site_directory_address := site_directory_address + 1;
              site_name              := master_directory[ site_directory_address ].site_name;

        until ( site_name = end_directory_string );

    end; { procedure calculate_lifestage_development }


procedure calculate_trap_catch_statistics;
    {
      This procedure calculates the following statistics : daily average count per
      trap, the daily total count over all traps, and the grand total count to
      date.  These statistics are calculated for each site which gathers trap data
      and for each pest which is monitored.
    }
    var
        pest_file_name         : line;
        site_name              : disk_file_name;
        full_site_name         : full_site_name_type;
        site_exists            : boolean;
        site_directory_address : integer;
        pest_type              : kinds_of_sites;


    procedure calculate_summary( pest_file_name : line );
        {
          This procedure calculates the summary statistics for the daily trap
          records of a specific site and pest.  The procedure stores the
          statistics in the disk file along with the original trap data.
        }
        type
            big_line = string[ 127 ];
        var
            trap_file                                       : text;
            dummy, first_record                             : big_line;
            new_record                                      : array[ 1..366 ] of big_line;
            i, middle_record, record_number, trap,
                  n_traps, max_traps, total, grand, count,
                  condition_code, start_of_grand_total,
                  grand_total_position, total_position      : integer;
    mean                                            : real;
    start_calculating                               : boolean;
    entry, average, total_today, number_of_traps,
          grand_total, missing_string               : line;
    date_str                                        : string[ width_of_date_field ];
    stats_str                                       : string[ width_of_stats_field ];
    trap_str                                        : big_line;
    trap_count                                      : array[ 1..10 ] of string[ width_of_trap_field ];

begin

    missing_string := 'MISSING';
    { Must ensure that sufficient leading spaces are supplied so that the
      word MISSING will fit the record field properly. }
    for i := 1 to ( width_of_average_field + width_of_todays_total_field + width_of_ntraps_field ) do
        missing_string := ' ' + missing_string;

    { assume the program does not have to immediately start calculating statistics }
    start_calculating := false;

    { starting column of the grand total sub-field within the statistics field }
    start_of_grand_total := 1 + width_of_stats_field - width_of_grand_total_field;

    { read old *.TRP file }
    assign( trap_file, pest_file_name );
    reset( trap_file );

    { The first record is used to calculate the maximum number of traps set out for this pest. }
    readln( trap_file, first_record );
    max_traps := round( ( length( first_record ) - ( width_of_date_field + width_of_stats_field ) ) /
                          width_of_trap_field );

    { set grand total to 0 and use as a dummy grand total field }
    str( 0 : width_of_grand_total_field, grand_total );

    record_number := 0;
    while not eof( trap_file ) do
        begin

            record_number := record_number + 1;

            { Each file record represents the data for one collection
              date.  The program assumes the number of traps monitored
              on any date is the maximum possible. }
            n_traps := max_traps;

            { can this be moved to the  if start_calculating block ??? }
            total   := 0;

            readln( trap_file, date_str, stats_str, trap_str );
            { note : the statistics field ( stats_str ) contains the
                     subfields : average, daily total and grand total to date }
```

```
{
  Start statistics calculations at the first trap record which
  has not been processed.  Such a record is indicated by a
  blank statistics field.

  Once statistics calculations have started they must continue
  to the end of the file even though statistics have been
  calculated for subsequent records.  This allows new records
  to be entered in any order; the statistics for all old
  records following a newly entered record will be recalculated.
  This also means that a "start_calculating" flag is required;
  the contents of the "stats_str" do not indicate if
  statistics calculations are necessary.
}

if   not start_calculating
then
     if   delete_blanks( stats_str ) = ''
     then
          start_calculating := true
     else
          { reconstruct the data record from the old data
            since the statistics field is OK }
          dummy := date_str + stats_str + trap_str;

{
  IMPORTANT

  The following "if start_calculating" statement appears to be
  the alternative ( else ] to the previous "if not
  start_calculating" statement ( as these statements are
  currently written ).  This is not the case because the
  previous "if" could set "start_calculating" to true
  implying that statistics must be calculated for the current
  record.  The seemingly correct "if-then-else" construction
  would set "start_calculating" to true and ignore the current
  record.
}
if   start_calculating
then
     begin

         for trap := 1 to max_traps do
             begin

                 { "i" equals the starting column of the trap
                    subfield for trap number "trap" within the
                    traps field of the data record. }
                 i := width_of_trap_field * ( trap - 1 ) + 1;
                 trap_count[ trap ] := copy( trap_str, i, width_of_trap_field );
```

```
                         entry := upper_case( delete_blanks( trap_count[ trap ] ) );
                         if   entry <> 'MISSING'
                         then
                              begin
                                  val( entry, count, condition_code );
                                  total := total + count;
                              end { if entry }
                         else { if an entry is MISSING then decrement number of traps }
                              n_traps := n_traps - 1;

                     end; { for i }

                 if   n_traps > 0
                 then
                      begin

                          mean := total / n_traps;
                          str( mean    : width_of_average_field : 1 , average );
                          str( total   : width_of_todays_total_field, total_today );
                          str( n_traps : width_of_ntraps_field      , number_of_traps );

                      end { if n_traps }
                 else
                      { all traps are missing }
                      begin

                          { indicate the summary statistics for this record are also missing }
                          average := last_characters_in_string( missing_string, width_of_average_field );
                          total_today := last_characters_in_string( missing_string, width_of_todays_total_field );
                          number_of_traps := last_characters_in_string( missing_string, width_of_ntraps_field );

                      end; { else if n_traps }

                 { construct data record }
                 dummy := date_str + average + total_today + number_of_traps + grand_total + trap_str;

             end; { if start_calculating }

        new_record[ record_number ] := dummy;

    end; { while }

close( trap_file );

{
  This section calculates the grand total fields in the list of
  trap records.  This is must be done separately and in reverse order
  because the trap records are stored in descending date order.  This
  section only needs to be called if some statistics must be recalculated.
}
if   start_calculating
then
```

```
        begin

            total_position       := width_of_date_field + width_of_average_field + 1;
            grand_total_position := width_of_date_field + start_of_grand_total;
            grand := 0;
            for i := record_number downto 1 do
                begin

                    entry := copy( new_record[ i ], total_position, width_of_todays_total_field );
                    entry := delete_blanks( entry );
                    val( entry, total, condition_code );
                    grand := grand + total;

                    str( grand : width_of_grand_total_field, grand_total );
                    delete( new_record[ i ], grand_total_position, width_of_grand_total_field );
                    insert( grand_total, new_record[ i ], grand_total_position );

                end; { for i }

        end; { if start_calculating }

    { Update *.TRP file. }

    assign( trap_file, pest_file_name );
    rewrite( trap_file );
    write( trap_file, first_record );

    for i := 1 to record_number do
        begin
            { End-of-file must occur at the end of a complete line.  Therefore,
              a null "writeln" precedes each "write" of a complete line. }
            writeln( trap_file );
            write( trap_file, new_record[ i ] );
        end; { for i }

    close( trap_file );

end; { procedure calculate_summary }


begin

    { start at the beginning of the list of sites }
    site_directory_address := 1;
    site_name := master_directory[ site_directory_address ].site_name;

    repeat

        { perform trap catch calculations at trap catch sites }
        pest_type := succ( first_pest );
```

```
             repeat

                  if   master_directory[ site_directory_address ].trapped[ pest_type ]
                  then
                       begin
                           pest_file_name := PMS_root_directory + Year_path + site_name +'\'+
                                             pest_file_name_vector[ pest_type ]+'.TRP';
                           calculate_summary( pest_file_name );
                       end; { if master_directory }

                  pest_type := succ( pest_type );

             until ( pest_type = last_pest );

             increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        until ( site_directory_address = 1 ); { stop when back at the beginning }

    end; { procedure calculate_trap_catch_statistics }


procedure calculations;
    {
      This procedure forms the "Calculate" task command.  It calculates the lifestage
      development using the heat unit models and it calculates the trap catch summary
      statistics.
    }
    begin

        writeln;
        writeln( ' Task : Calculations' );
        writeln;

        calculate_lifestage_development;
        calculate_trap_catch_statistics;

    end; { procedure calculations }
```

```
PRINTREP.PAS
------------

procedure print_report;
    {
      This procedure prints a set of reports for each requested site.
      There are four possible reports :
            1. a trap catch summary - short form
            2. a trap catch summary - full season
            3. a list of pests and the predicted development
                 of current lifestage(s)
            4. a complete list of population events as from report 2.
    }
    const
        indent    = '      '; { indentation spaces for the lifestage names }
        width_of_pest_name        = 26;
        width_of_output_lifestage = 30;
    type
        output_device_type = ( no_device, screen, printer );

        {
          Print requests are stored in a dynamic variable because the number
          of requests is dynamic.  Also, dynamic variables are stored in the
          heap instead of the data segment.  This allows the list of requests
          to grow quite large and reduces the demand on the data segment.

          The number of requests can exceed the number of sites because the
          user may place several requests for a specific printout, i.e. copies.
        }
        request_pointer    = ^request_entry;
        request_entry      = record
                                directory_address : integer;
                                want_trap_report_short, want_trap_report_full_season,
                                      want_lifestage_report, want_events_report : boolean;
                                output_device : output_device_type;
                                next : request_pointer;
                             end; { record }

    var
        top_of_heap : ^integer;
        first_entry, last_entry, new_entry : request_pointer;

        response            : line;
        output_file         : text;
        output_logical_file : output_device_type;


    procedure get_yes_no_reply( prompt : line; var answer : line );
        var
            ans : line;

        begin
        repeat
            write( ' Do you want ' + prompt + ' ? ' );
            readln( ans );
            ans := upper_case( delete_blanks( ans ) );
            if   ans = ''    { null answer not allowed }
            then ans := ' ';
            if   not ( ans[ 1 ] in [ 'Y', 'N' ] )
            then writeln( ' Enter either YES or NO.' );
        until ( ans[ 1 ] in [ 'Y', 'N' ] );

        answer := ans;

    end; { procedure get_yes_no_reply }


procedure fetch_wanted_reports( var want_trap_short, want_trap_full, want_lifestage, want_events : boolean );
    {
      This procedure prompts the user for which reports are desired.
    }
    var
        prompt, ans : line;

    begin

        get_yes_no_reply( 'trap catch summaries - short form', ans );
        want_trap_short := ( ans[ 1 ] = 'Y' );

        get_yes_no_reply( 'trap catch summaries - full season', ans );
        want_trap_full := ( ans[ 1 ] = 'Y' );

        get_yes_no_reply( 'lifestage development reports', ans );
        want_lifestage := ( ans[ 1 ] = 'Y' );

        get_yes_no_reply( 'an events report', ans );
        want_events := ( ans[ 1 ] = 'Y' );

    end; { procedure fetch_wanted_reports }


procedure construct_list_of_all_sites;
    {
      This procedure constructs a list consisting of all of the sites in the
      MASTER directory.  The user is asked which reports to print.  The same
      set of reports is printed for each site.  For this reporting option
      only one copy of each report and only hardcopy printout is allowed.
    }
    var
        site_name              : disk_file_name;
        full_site_name         : full_site_name_type;
        site_exists            : boolean;
        site_directory_address : integer;
        want_trap_short, want_trap_full, want_lifestage, want_events : boolean;

    begin

        fetch_wanted_reports( want_trap_short, want_trap_full, want_lifestage, want_events );

        { start at the beginning of the list of sites }
        site_directory_address := 0;
        increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        repeat

            new( new_entry ); { allocate new variable }

            new_entry^.directory_address            := site_directory_address;
            new_entry^.want_trap_report_short       := want_trap_short;
            new_entry^.want_trap_report_full_season := want_trap_full;
            new_entry^.want_lifestage_report        := want_lifestage;
            new_entry^.want_events_report           := want_events;
            new_entry^.output_device                := printer;

            if   first_entry = nil
            then first_entry := new_entry
            else last_entry^.next := new_entry;

            last_entry := new_entry;
            last_entry^.next := nil;

            increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        until ( site_directory_address = 1 ); { stop when back to the beginning }

    end; { procedure construct_list_of_all_sites }


procedure construct_list_of_requests;
    {
      This procedure is more flexible than procedure construct_list_of_all_sites.
      It allows the user to completely control which reports are produced.

      The procedure prompts the user for site names and asks the user which
      report(s) to print and on which output device ( screen or printer ).
      Site names may be entered in any order.  This reporting option allows
      multiple requests for the same report, i.e. copies.
    }
    var
        ans, response                       : line;
        site_name                           : disk_file_name;
        full_site_name                      : full_site_name_type;
        site_exists                         : boolean;
        site_directory_address              : integer;
        want_trap_short, want_trap_full, want_lifestage, want_events : boolean;
    begin

        site_directory_address := 0;
        increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        repeat

             fetch_site_name( '', response, site_name, full_site_name, site_exists, site_directory_address );

             if   response <> nomore
             then
                  begin
                      new( new_entry ); { allocate new variable }

                      fetch_wanted_reports( want_trap_short, want_trap_full, want_lifestage, want_events );

                      get_yes_no_reply( 'the report(s) sent to the printer', ans );
                      if   ( ans[ 1 ] = 'Y' )
                      then new_entry^.output_device := printer
                      else
                           begin
                               new_entry^.output_device := screen;
                               writeln( ' Report(s) will appear on the screen.' );
                           end;

                      new_entry^.directory_address            := site_directory_address;
                      new_entry^.want_trap_report_short       := want_trap_short;
                      new_entry^.want_trap_report_full_season := want_trap_full;
                      new_entry^.want_lifestage_report        := want_lifestage;
                      new_entry^.want_events_report           := want_events;

                      if   first_entry = nil
                      then first_entry := new_entry
                      else last_entry^.next := new_entry;

                      last_entry := new_entry;
                      last_entry^.next := nil;

                  end; { if response }

             increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        until ( response = nomore );

    end; { procedure construct_list_of_requests }


procedure report_header( site_name : disk_file_name; full_site_name : full_site_name_type;
                         nearest_weather_site : disk_file_name; region : region_name_type );
    {
      This procedure produces a common header for all reports.
      The date and year are passed as global variables.

      If an input parameter is the null string, the corresponding label in the
      header is suppressed.
    }
    var
        this_date : string[ 6 ];
    begin

        this_date := date_string_from_Julian( todays_Julian_date );

        writeln( output_file );

        if   full_site_name <> ''
        then writeln( output_file, ' Site                 : ', full_site_name );

        if   site_name <> ''
        then writeln( output_file, ' Site Abbreviation    : ', site_name );

        if   region <> ''
        then writeln( output_file, ' Region               : ', region );

        if   nearest_weather_site <> ''
        then writeln( output_file, ' Nearest Weather Site : ', nearest_weather_site );

        writeln( output_file, ' Season               : ', this_year : 4 );
        writeln( output_file, ' Today                : ', this_date : 6 );
        writeln( output_file );

    end; { procedure report_header }


procedure report_trailer;
    {
      This procedure produces a common trailer for all reports.
    }
    var
        dummy : line;

    begin

        writeln( output_file );
        writeln( output_file );

        if   output_logical_file = screen
        then
             { Give the user a chance to read the screen and allow her to
               decide when to continue to the next one. }
             begin
                 writeln;
                 write( ' press RETURN to continue' );
                 readln( dummy );
        end;

      writeln( output_file, form_feed );

    end; { procedure report_trailer }


procedure trap_catch_report( site_name : disk_file_name; full_site_name : full_site_name_type;
                             nearest_weather_site : disk_file_name; region : region_name_type;
                             site_directory_address : integer; want_full_season : boolean );
    {
      This procedure produces the body of a trap catch report.
    }
    const
      { Note : These constants are candidates for inclusion in BUGSYS.PAR. }

      { widths of output fields }
      width_of_date        =  8;
      width_of_average     =  8;
      width_of_total       =  8;
      width_of_ntraps      =  8;
      width_of_grand_total = 12;

      { maximum number of daily trap count summaries in a short form }
      lines_in_short_form  =  2;

    var
      dummy, path_name, input_file_name,
                    trailing_spaces, pest_name : line;
      first_file, trapped_pests, some_trapped : boolean;
      i               : integer;
      lines_in_form   : integer;
      input_file_var  : text;
      date            : numeric_date_string_type;
      date_str        : string[ 6 ];
      average         : string[ width_of_average_field ];
      total_today     : string[ width_of_todays_total_field ];
      number_of_traps : string[ width_of_ntraps_field ];
      grand_total     : string[ width_of_grand_total_field ];
      pest_type       : kinds_of_sites;

    begin

      trailing_spaces := '';
      for i := 1 to width_of_pest_name do trailing_spaces := trailing_spaces + ' ';

      writeln( output_file );
      writeln( output_file, ' Report of Trap Catches' );
      writeln( output_file, ' ======================' );

      report_header( site_name, full_site_name, nearest_weather_site, region );
```

```
path_name := PMS_root_directory + Year_path + site_name + '\';

first_file    := true;
trapped_pests := false; { assume no pests are trapped }

for  pest_type := succ( first_pest ) to pred( last_pest ) do
     begin

         if   master_directory[ site_directory_address ].trapped[ pest_type ]
         then
              begin

                  trapped_pests := true;  { at least one pest is trapped }
                  some_trapped  := false; { assume no trap catches have been recorded yet }

                  input_file_name := path_name + pest_file_name_vector[ pest_type ] + '.TRP';
                  pest_name       := pest_name_vector[ pest_type ] + trailing_spaces;
                  pest_name       := copy( pest_name, 1, width_of_pest_name );

                  assign( input_file_var, input_file_name );
                  reset( input_file_var );

                  if   first_file
                  then
                       {
                         Note : The boolean flag "first_file" is set to FALSE the first time this
                                block is executed.  Thus, this block is executed just once, and so the
                                column headers are printed just once.
                       }
                       begin

                           first_file := false;

                           { Place trailing spaces after 'Trapped Pest' and cut string to proper length.
                             This ensures 'Trapped Pest' is left justified in the output field. }
                           dummy := 'Trapped Pest' + trailing_spaces;
                           dummy := copy( dummy, 1, width_of_pest_name );

                           writeln( output_file, ' ', dummy : width_of_pest_name, 'Date' : width_of_date,
                                                  'Average' : width_of_average, 'Total' : width_of_total,
                                                  'Traps' : width_of_ntraps, 'Grand Total' : width_of_grand_total );
                           writeln( output_file );
                           writeln( output_file );

                       end; { first_file }

                  readln( input_file_var ); { read dummy first line and ignore it }

                  { If the end-of-file occurs immediately after the first dummy line,
                    then no trap catches have been recorded to date. }
                  if   eof( input_file_var )
                  then
```

```
                               begin
                                   writeln( output_file, ' ', pest_name : width_of_pest_name,
                                                  ' No trap catches have been recorded yet.' );
                                   writeln( output_file );
                               end; { if eof( input_file_var ) }

                          {
                            The trap catch report prints summary statistics starting with the latest
                            and ending with the earliest.
                          }

                          if   want_full_season
                          then lines_in_form := 366                    { up to a full year }
                          else lines_in_form := lines_in_short_form;

                          i := 1; { counter for number of daily trap counts written to report }
                          while ( ( not eof( input_file_var ) ) and ( i <= lines_in_form ) ) do
                              begin

                                  readln( input_file_var, date, average, total_today,
                                                 number_of_traps, grand_total );

                                  date_str := date_string_from_numeric( date );

                                  writeln( output_file, ' ', pest_name : width_of_pest_name,
                                             date_str : width_of_date, average : width_of_average,
                                             total_today : width_of_total, number_of_traps : width_of_ntraps,
                                             grand_total : width_of_grand_total );

                                  i := i + 1;

                              end; { while ( ( not eof( input_file_var ) ) and ( i <= lines_in_form ) ) }

                          writeln( output_file );

                          close( input_file_var );

                      end; { if master_directory }

             end; { for pest_type }

        if   not trapped_pests
        then writeln( output_file, ' No pests are trapped at this site.' );

        report_trailer;

    end; { procedure trap_catch_report }


procedure lifestage_development_report( site_name : disk_file_name; full_site_name : full_site_name_type;
                                        nearest_weather_site : disk_file_name;  region : region_name_type;
                                        site_directory_address : integer );
```

```
{
  This procedure produces the body of a lifestage development report.
}
const
    width_of_output_development = 10;
var
    dummy, path_name, input_file_name, pest_name,
            pest_spaces                                     : line;
    input_file_var                                          : text;
    i                                                       : integer;
    active_char                                             : char;
    pest_type                                               : kinds_of_sites;
    generation                                              : string[ 2 ];
    lifestage_name                                          : string[ 30 ];
    pest_activity, first_file, modelled_pests               : boolean;
    Julian_day_of_last_actual, Julian_day_of_forecast       : integer;
    development_last_actual, percent_last_actual,
            development_forecast, percent_forecast          : real;
    date_str_last, date_str_forecast                        : string[ 6 ];


begin

    pest_spaces := '';
    for i := 1 to width_of_pest_name do pest_spaces := pest_spaces + ' ';

    writeln( output_file );
    writeln( output_file, ' Report of Lifestage Development' );
    writeln( output_file, ' ===============================' );

    report_header( site_name, full_site_name, nearest_weather_site, region );

    path_name := PMS_root_directory + Year_path + nearest_weather_site + '\';

    first_file     := true;
    modelled_pests := false; { assume no pests are modelled }

    for  pest_type := succ( first_pest ) to pred( last_pest ) do
         begin

             pest_activity := false; { assume no pests have any active lifestages }

             if   master_directory[ site_directory_address ].modelled[ pest_type ]
             then
                  begin

                      modelled_pests  := true; { at least one pest is modelled }

                      input_file_name := path_name + pest_file_name_vector[ pest_type ] + '.MDL';
                      pest_name       := pest_name_vector[ pest_type ] + pest_spaces;
                      pest_name       := copy( pest_name, 1, width_of_pest_name );
```

```
assign( input_file_var, input_file_name );
reset( input_file_var );

if   first_file
then
     begin

         {
           Note : The boolean flag "first_file" is set to FALSE the
                  first time this block is executed.  Thus this block is
                  executed just once, and so the column headers are printed
                  just once.
         }
         first_file := false;

         {
           There is no dummy first line in a *.MDL file.  Therefore, the first
           line must be read, the dates extracted and then the file pointer
           must be returned to the beginning of the file.
         }
         readln( input_file_var, active_char, generation, lifestage_name,
                      Julian_day_of_last_actual, development_last_actual,
                      Julian_day_of_forecast, development_forecast );
         reset( input_file_var );

         date_str_last  := date_string_from_Julian( Julian_day_of_last_actual);
         date_str_forecast := date_string_from_Julian( Julian_day_of_forecast);

         writeln( output_file );
         writeln( output_file, indent, ' ' : width_of_output_lifestage,
                     'Percent Development' : width_of_output_development + 10 );
         writeln( output_file );
         writeln( output_file, indent, ' ' : width_of_output_lifestage,
                             date_str_last : width_of_output_development,
                             date_str_forecast : width_of_output_development );
         writeln( output_file );
         writeln( output_file );
         writeln( output_file, ' ', pest_name : width_of_pest_name );
         writeln( output_file );

     end { if first_file }

else
     begin
         writeln( output_file, ' ', pest_name : width_of_pest_name );
         writeln( output_file );
     end;

while not eof( input_file_var ) do
    begin

        read( input_file_var, active_char );
```

```
                                   if   ( active_char = 'A' )  { currently active stage }
                                   then
                                        begin

                                            pest_activity := true;

                                            readln( input_file_var, generation, lifestage_name, Julian_day_of_last_actual,
                                                   development_last_actual, Julian_day_of_forecast, development_forecast );

                                            percent_last_actual := 100.0 * development_last_actual; { convert to percent }
                                            percent_forecast    := 100.0 * development_forecast;

                                            writeln( output_file, indent, lifestage_name : width_of_output_lifestage,
                                                                  percent_last_actual : width_of_output_development : 1,
                                                                  percent_forecast : width_of_output_development : 1 );
                                            writeln( output_file );

                                        end { end if active_char }
                                   else
                                        readln( input_file_var ); { skip to next line }

                               end; { while }

                          close( input_file_var );

                          if   not pest_activity
                          then writeln( output_file, indent, ' No current activity' );

                          writeln( output_file );
                          writeln( output_file );

                      end; { if master_directory }

             end; { for pest_type }

        if   not modelled_pests
        then writeln( output_file, ' No pests are modelled at this site.' );

        report_trailer;

    end; { procedure lifestage_development_report }


procedure population_events_report( site_name : disk_file_name; full_site_name : full_site_name_type;
                                    nearest_weather_site : disk_file_name; region : region_name_type;
                                    site_directory_address : integer );
    {
      This procedure produces the body of a population events report.  The
      report consists of the dates on which specific lifestages completed
      development.  Therefore, currently active and hibernating lifestages
      are not included in the report.
    }
```

```
const
    width_of_discard_field      = 32; { 4 + 8 + 4 + 8 + 4 + 4 }
    width_of_date_column_field  = 16;
var
    dummy, path_name, input_file_name, pest_name, pest_spaces  : line;
    completely_developed_lifestage, first_file, modelled_pests : boolean;
    input_file_var   : text;
    i                : integer;
    active_char      : char;
    pest_type        : kinds_of_sites;
    discard          : string[ width_of_discard_field ];
    generation       : string[ 2 ];
    lifestage_name   : string[ 30 ];
    date_100         : integer;
    date_str_100     : string[ 6 ];


begin

    pest_spaces := '';
    for i := 1 to width_of_pest_name do pest_spaces := pest_spaces + ' ';

    writeln( output_file );
    writeln( output_file, ' Report of Population Events' );
    writeln( output_file, ' ===========================' );

    report_header( site_name, full_site_name, nearest_weather_site, region );

    path_name := PMS_root_directory + Year_path + nearest_weather_site + '\';

    first_file     := true;
    modelled_pests := false; { assume no pests are modelled }

    for  pest_type := succ( first_pest ) to pred( last_pest ) do
         begin

              completely_developed_lifestage := false; { assume no lifestages have completed development }

              if   master_directory[ site_directory_address ].modelled[ pest_type ]
              then
                   begin

                        modelled_pests := true; { at least one pest is modelled }

                        input_file_name := path_name + pest_file_name_vector[ pest_type ] + '.MDL';
                        pest_name       := pest_name_vector[ pest_type ] + pest_spaces;
                        pest_name       := copy( pest_name, 1, width_of_pest_name );

                        assign( input_file_var, input_file_name );
                        reset( input_file_var );

                        if   first_file
```

```
then
     begin

         {
           Note : The boolean flag "first_file" is set to FALSE the first time
                  this block is executed.  Thus, this block is executed just once,
                  and so the column headers are printed just once.
         }
         first_file := false;

         writeln( output_file );
         writeln( output_file, indent, ' ' : width_of_output_lifestage,
                                 'Predicted' : width_of_date_column_field );
         writeln( output_file );
         writeln( output_file );

     end; { if first_file }


writeln( output_file, ' ', pest_name : width_of_pest_name );
writeln( output_file );

while not eof( input_file_var ) do
    begin

        read( input_file_var, active_char );
        if   ( active_char = 'P' )  { completely developed lifestage }
        then
             begin

                  completely_developed_lifestage := true;

                  readln( input_file_var, generation, lifestage_name, discard, date_100 );

                  date_str_100 := date_string_from_Julian( date_100 );

                  writeln( output_file, indent, lifestage_name : width_of_output_lifestage,
                                         date_str_100 : width_of_date_column_field );
                  writeln( output_file );

             end { end if active_char }
        else
             readln( input_file_var ); { skip to next line }

    end; { while }

close( input_file_var );

if   not completely_developed_lifestage
then writeln( output_file, indent, ' No lifestages have completed development.' );

writeln( output_file );
```

```
                        writeln( output_file );

                    end; { if master_directory }

                end; { for pest_type }

            if   not modelled_pests
            then writeln( output_file, ' No pests are modelled at this site.' );

            report_trailer;

        end; { procedure population_events_report }


    procedure print_list_of_reports;
        {
          This procedure reads the list of requested reports and prints them.
        }
        var
            site_name, nearest_weather_site  : disk_file_name;
            full_site_name : full_site_name_type;
            region : region_name_type;

        begin

            while first_entry <> nil do
                  with first_entry^ do
                       begin

                           full_site_name       := master_directory[ directory_address ].full_name;
                           site_name            := master_directory[ directory_address ].site_name;
                           nearest_weather_site := master_directory[ directory_address ].nearest_weather_site;
                           region               := master_directory[ directory_address ].region;

                           { output device }
                           output_logical_file := output_device;
                           if   output_device = printer
                           then assign( output_file, 'LST:' )
                           else assign( output_file, 'CON:' ); { screen }
                           rewrite( output_file );

                           if   want_trap_report_short
                           then trap_catch_report( site_name, full_site_name, nearest_weather_site,
                                                   region, directory_address, false );

                           if   want_trap_report_full_season
                           then trap_catch_report( site_name, full_site_name, nearest_weather_site,
                                                   region, directory_address, true );

                           if   want_lifestage_report
                           then lifestage_development_report( site_name, full_site_name, nearest_weather_site,
                                                   region, directory_address );
                        if   want_events_report
                        then population_events_report( site_name, full_site_name, nearest_weather_site,
                                                           region, directory_address );

                        first_entry := next;

                        close( output_file ); { no function on logical devices }

                    end; { while first_entry }

    end; { procedure print_list_of_reports }


begin

    writeln;
    writeln( ' Task : Print reports' );
    writeln;

    mark( top_of_heap );
    first_entry := nil;

    get_yes_no_reply( 'all reports for all the sites', response );
    if   response[ 1 ] = 'Y'
    then construct_list_of_all_sites
    else construct_list_of_requests;

    print_list_of_reports;

    release( top_of_heap );

end; { procedure print_report }
```

```
REGIONAL.PAS
------------

procedure print_regional_reports;
    {
      This procedure prints a set of regional reports.  The reports are compilations
      of lifestage development for each site within the region.

      Note : 1. The current version produces reports for each region
                listed in the master directory; there is no user selection.
    }

    const
      width_of_entry_field = 7;
      separator            = '|'; { for separating the fields from one site to the next }

    type
      output_device_type = ( screen, printer );
      big_line = string[ 255 ];

      {
        Print requests are stored in a dynamic variable because the number
        of requests is dynamic.  Also, dynamic variables are stored in the
        heap instead of the data segment.  This allows the list of requests
        to grow quite large and reduces the demand on the data segment.

        The number of requests can exceed the number of sites because the
        user may place several requests for a specific printout, i.e. copies.
      }
      request_pointer    = ^request_entry;
      request_entry      = record
                             region_name                      : region_name_type;
                             start_region_directory_address,
                             end_region_directory_address     : integer;
                             output_device                    : output_device_type;
                             next                             : request_pointer;
                           end; { record }

      pest_stages = record
                          pest_name : big_line;
                          pest_stage_name : array[ 1..max_pest_stages ] of big_line;
                          n_pest_stages   : integer;
                    end; { record }

    var
      top_of_heap : ^integer;
      first_entry, last_entry, new_entry : request_pointer;

      sites_per_regional_report_page : integer;

      response          : line;
      output_file       : text;
    output_logical_file : output_device_type;

    blank_site_name_line, site_name_line : array[ 1..4 ] of big_line;
    blank_dates_line, dates_line         : big_line;
    blank_report_line, report_line       : array[ first_pest .. last_pest ] of pest_stages;
    theoretical_model_exists             : array[ first_pest .. last_pest ] of boolean;


procedure get_yes_no_reply( prompt : line; var answer : line );
    var
        ans : line;

    begin

        repeat
            write( ' Do you want ' + prompt + ' ? ' );
            readln( ans );
            ans := upper_case( delete_blanks( ans ) );
            if   ans = ''    { null answer not allowed }
            then ans := ' ';
            if   not ( ans[ 1 ] in [ 'Y', 'N' ] )
            then writeln( ' Enter either YES or NO.' );
        until ( ans[ 1 ] in [ 'Y', 'N' ] );

        answer := ans;

    end; { procedure get_yes_no_reply }


procedure advance_to_next_region( var region_B : region_name_type;
                                  var site_name : disk_file_name; var full_site_name : full_site_name_type;
                                  var site_exists : boolean; var site_directory_address : integer );
    var
        name_A, name_B : region_name_type;
    begin

        name_A := upper_case( delete_blanks( region_B ) );

        repeat
            increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );
            region_B := master_directory[ site_directory_address ].region;
            name_B   := upper_case( delete_blanks( region_B ) );
        until ( name_B <> name_A );

    end; { procedure advance_to_next_region }


procedure construct_list_of_all_regions;
    {
      This procedure constructs a list consisting of all of the regions in the
      MASTER directory.  The same report is printed for each region.  This
      reporting option allows only one hardcopy of the report.
    }
    var
        site_name              : disk_file_name;
        full_site_name         : full_site_name_type;
        site_exists            : boolean;
        site_directory_address : integer;
        region_name_B          : region_name_type;

    begin

        { start at the beginning of the list of sites }
        site_directory_address := 1;
        region_name_B          := master_directory[ site_directory_address ].region;

        repeat

             new( new_entry ); { allocate new variable }

             new_entry^.region_name                  := region_name_B;
             new_entry^.start_region_directory_address := site_directory_address;
             new_entry^.output_device                := printer;

             if   first_entry = nil
             then first_entry := new_entry
             else last_entry^.next := new_entry;

             last_entry := new_entry;
             last_entry^.next := nil;

             advance_to_next_region( region_name_B, site_name, full_site_name,
                                     site_exists, site_directory_address );

             if   site_directory_address <> 1
             then new_entry^.end_region_directory_address := site_directory_address - 1
             else new_entry^.end_region_directory_address := max_master_directory_index - 1;

        until ( site_directory_address = 1 ); { stop when back to the beginning }

    end; { procedure construct_list_of_all_regions }


procedure construct_list_of_requests;
    {
      This procedure prompts the user for region names and on which output
      device ( screen or printer ) she wants the output directed.
      Region names may be entered in any order.  This reporting option allows
      multiple requests for the same report, i.e. copies.
    }
    var
        ans, response, prompt        : line;
        site_name              : disk_file_name;
        full_site_name         : full_site_name_type;
    site_exists            : boolean;
    site_directory_address : integer;
    region_name_B          : region_name_type;

begin

    { start at the beginning of the list of sites }
    site_directory_address := 1;
    region_name_B          := master_directory[ site_directory_address ].region;

    repeat

         get_yes_no_reply( 'a report for ' + region_name_B, ans );

         if   ans[ 1 ] = 'Y'
         then
              begin

                  new( new_entry ); { allocate new variable }

                  new_entry^.start_region_directory_address := site_directory_address;

                  get_yes_no_reply( 'the report(s) sent to the printer', response );
                  if   ( response[ 1 ] = 'Y' )
                  then new_entry^.output_device := printer
                  else
                       begin
                           new_entry^.output_device := screen;
                           writeln( ' Report(s) will appear on the screen.' );
                       end;

                  if   first_entry = nil
                  then first_entry := new_entry
                  else last_entry^.next := new_entry;

                  last_entry := new_entry;
                  last_entry^.next := nil;

                  advance_to_next_region( region_name_B, site_name, full_site_name,
                                          site_exists, site_directory_address );

                  if   site_directory_address <> 1
                  then new_entry^.end_region_directory_address := site_directory_address - 1
                  else new_entry^.end_region_directory_address := max_master_directory_index - 1;

              end
         else
              begin
                  advance_to_next_region( region_name_B, site_name, full_site_name,
                                          site_exists, site_directory_address );
              end; { if ans[ 1 ] }
```

```
            get_yes_no_reply( 'a report for another region', response );

        until ( response[ 1 ] = 'N' );

    end; { procedure construct_list_of_requests }


procedure make_line_prefixes;
    {
      This procedure constructs the prefixes for the individual lifestages reported
      in the regional report.
    }
    var
        pest_type           : kinds_of_sites;
        i, n_lifestages     : integer;
        RAT_file, MDL_file  : text;
        dummy_3             : string[ 3 ];
        indent              : string[ 10 ];
        stage_name          : string[ width_of_stage_name_field ];
        path_name, file_name, pest_name, leading_blanks, dummy : line;

    begin

        path_name := PMS_root_directory + Year_path + 'INITSITE\';
        indent    := '          ';

        leading_blanks := '';
        for i := 1 to length( indent ) + width_of_stage_name_field do leading_blanks := leading_blanks + ' ';

        for i := 1 to 4 do blank_site_name_line[ i ] := leading_blanks + separator;
        blank_dates_line := leading_blanks + separator;

        for pest_type := succ( first_pest ) to pred( last_pest ) do
            begin

                file_name := path_name + pest_file_name_vector[ pest_type ] + '.RAT';

                if   file_exist( file_name )
                then
                     begin

                          theoretical_model_exists[ pest_type ] := true;

                          { read *.RAT file to get rate matrix }
                          assign( RAT_file, file_name );
                          reset( RAT_file );

                          readln( RAT_file, dummy ); { first line used to determine number of lifestages }
                          { Note : The rate index field is the temperature field. }
                          n_lifestages := round( ( length( dummy ) - ( width_of_rate_index_field ) ) /
                                                      width_of_rate_field );
                          close( RAT_file );

                          with blank_report_line[ pest_type ] do
                               begin

                                   pest_name := pest_name_vector[ pest_type ] + leading_blanks + separator;
                                   pest_name := copy( pest_name, 1, length( leading_blanks + separator ) );

                                   file_name := path_name + pest_file_name_vector[ pest_type ] + '.MDL';
                                   assign( MDL_file, file_name );
                                   reset( MDL_file );

                                   for i := 1 to n_lifestages do
                                       begin
                                           readln( MDL_file, dummy_3, stage_name );
                                           pest_stage_name[ i ] := indent + stage_name + separator;
                                       end; { for i := 1 to n_lifestages }

                                   close( MDL_file );

                                   n_pest_stages := n_lifestages;

                               end; { with blank_report_line[ pest_type ] }

                     end
                else theoretical_model_exists[ pest_type ] := false;
                { if file_exist }

            end; { for pest_type }

    end; { procedure make_line_prefixes }


procedure region_report_header( region : region_name_type );
    {
      This procedure produces a common header for all reports.
      The date and year are passed as global variables.
    }
    var
        this_date : string[ 6 ];
    begin

        this_date := date_string_from_Julian( todays_Julian_date );

        writeln( output_file );
        writeln( output_file, ' Percent Development' );
        writeln( output_file );

        if   region <> ''
        then writeln( output_file, ' Region : ', region );

        writeln( output_file, ' Season : ', this_year : 4 );
        writeln( output_file, ' Today  : ', this_date : 6 );
        writeln( output_file );

    end; { procedure region_report_header }


procedure region_report_trailer;
    {
      This procedure produces a common trailer for all reports.
    }
    const
        form_feed = #12;
    var
        dummy : line;

    begin

        writeln( output_file );
        writeln( output_file );

        if   output_logical_file = screen
        then
             { Give the user a chance to read the screen and allow her to
               decide when to continue to the next one. }
             begin
                 writeln;
                 write( ' press RETURN to continue' );
                 readln( dummy );
             end;

        writeln( output_file, form_feed );

    end; { procedure region_report_trailer }


procedure regional_development_report( start_region_directory_address, end_region_directory_address : integer );
    {
      This procedure produces the body of a regional lifestage development report.
    }
    var
        site_file_name, path_name, site_path_name,
                file_name, blanks, entry1, entry2,
                dummy                                : line;
        MDL_file                                     : text;
        i, lifestage                                 : integer;
        active_flag                                  : char;
        pest_type                                    : kinds_of_sites;
        generation                                   : string[ 2 ];
        lifestage_name                               : string[ width_of_stage_name_field ];
        development_actual, development_forecast,
                percent_actual, percent_forecast     : real;
    { dates on which certain events occur }
    date_000_modelled, date_000_observed,              { date on which   0% development occurs ( start date ) }
    date_100_modelled, date_100_observed,              { date on which 100% development occurs ( end date )   }
    date_end_active_modelled, date_end_active_observed, { end of the active lifestage }
    date_actual, date_forecast                         { most recent date processed or forecast }
                                             : integer;

    string_date_actual, string_date_forecast   : string[ width_of_entry_field ];

begin

    blanks := '';
    for i := 1 to width_of_entry_field do blanks := blanks + ' ';

    path_name := PMS_root_directory + Year_path;

    for i := 1 to 4 do site_name_line[ i ] := blank_site_name_line[ i ];
    dates_line  := blank_dates_line;
    report_line := blank_report_line;

    for i := start_region_directory_address to end_region_directory_address do
        begin

            site_file_name := master_directory[ i ].site_name;
            site_path_name := path_name + site_file_name + '\';
            dummy := copy( ' ' + site_file_name + blanks + blanks, 1, 2 * width_of_entry_field );
            site_name_line[ 1 ] := site_name_line[ 1 ] + dummy + separator;
            { for now, do not use the remaining site name lines }
            site_name_line[ 2 ] := site_name_line[ 2 ] + blanks + blanks + separator;
            site_name_line[ 3 ] := site_name_line[ 3 ] + blanks + blanks + separator;
            site_name_line[ 4 ] := site_name_line[ 4 ] + blanks + blanks + separator;

            string_date_actual   := blanks;
            string_date_forecast := blanks;

            for pest_type := succ( first_pest ) to pred( last_pest ) do
                if  theoretical_model_exists[ pest_type ]
                then

                    if   master_directory[ i ].modelled[ pest_type ]
                    then
                         begin

                             file_name := site_path_name + pest_file_name_vector[ pest_type ] + '.MDL';
                             assign( MDL_file, file_name );
                             reset( MDL_file );

                             with report_line[ pest_type ] do
                                  begin

                                      pest_name := pest_name + blanks + blanks + separator;
```

```
for lifestage := 1 to n_pest_stages do
    begin

        readln( MOL_file,
                active_flag, generation,
                lifestage_name,
                date_actual, development_actual,
                date_forecast, development_forecast,
                date_000_modelled, date_000_observed,
                date_100_modelled, date_100_observed,
                date_end_active_modelled, date_end_active_observed );

        case active_flag of
             'P' : begin
                       entry1 := 'PAST';
                       entry2 := 'PAST';
                   end;
             'A' : begin
                     { construct percent fields }
                     percent_actual := 100.0 * development_actual;
                     str( percent_actual : width_of_entry_field : 1, entry1 );
                     percent_forecast := 100.0 * development_forecast;
                     str( percent_forecast : width_of_entry_field : 1, entry2 );

                     { construct dates line }
                     dummy := blanks +
                                date_string_from_Julian( date_actual ) + ' ';
                     string_date_actual :=last_characters_in_string(
                                              dummy, width_of_entry_field );
                     dummy := blanks +
                                date_string_from_Julian( date_forecast ) + ' ';
                     string_date_forecast :=last_characters_in_string(
                                              dummy, width_of_entry_field );
                  end;
             'H' : begin
                       entry1 := ' ';
                       entry2 := ' ';
                   end;
            else  begin
                       entry1 := 'xxx';
                       entry2 := 'xxx';
                   end;
        end; { case }

        entry1 := blanks + entry1 + ' '; { add padding blanks };
        entry1 := last_characters_in_string( entry1, width_of_entry_field );
        entry2 := blanks + entry2 + ' '; { add padding blanks };
        entry2 := last_characters_in_string( entry2, width_of_entry_field );

        pest_stage_name[ lifestage ] := pest_stage_name[ lifestage ] +
                                        entry1 + entry2 + separator;
```

```
                                               end; { for lifestage := 1 to n_pest_stages }

                                           close( MDL_file );

                                       end; { with }

                              end
                         else
                              begin

                                  with report_line[ pest_type ] do
                                       begin
                                           entry1    := blanks + 'NOT MODELLED ';
                                           entry1    := last_characters_in_string( entry1, 2*width_of_entry_field );
                                           pest_name := pest_name + entry1 + separator;

                                           { blank fields }
                                           for lifestage := 1 to n_pest_stages do
                                               pest_stage_name[ lifestage ] := pest_stage_name[ lifestage ] +
                                                                               blanks + blanks + separator;
                                       end; { with }

                              end; { if master_directory[ i ].modelled[ pest_type ] }

                already_line := dates_line + string_date_actual + string_date_forecast + separator;

            end; { for i }

        { print report }
        for i := 1 to 4 do writeln( output_file, site_name_line[ i ] );
        writeln( output_file );
        writeln( output_file, dates_line );
        for pest_type := succ( first_pest ) to pred( last_pest ) do
           if  theoretical_model_exists[ pest_type ]
           then
                begin
                    writeln( output_file );
                    with report_line[ pest_type ] do
                         begin
                             writeln( output_file, pest_name );
                             for i := 1 to n_pest_stages do writeln( output_file, pest_stage_name[ i ] );
                         end; { with }
                end; { if theoretical_model_exists[ pest_type ] }

    end; { procedure regional_development_report }


procedure print_list_of_regional_reports;
    {
      This procedure reads the list of requested regional reports and prints them.
    }
    var
    region_name                      : line;
    i, start_address, end_address : integer;
    pest_type                        : kinds_of_sites;
    total_columns                    : integer; { total number of columns on one page of the report }
begin

    while first_entry <> nil do
          with first_entry^ do
               begin

                    region_name := master_directory[ start_region_directory_address ].region;

                    { output device }
                    output_logical_file := output_device;
                    if   output_device = printer
                    then
                         begin
                              assign( output_file, 'LST:' );
                              total_columns := ( trunc( width_of_paper ) ) * characters_per_inch;
                         end
                    else
                         begin
                              assign( output_file, 'CON:' ); { screen }
                              total_columns := 80;
                         end;
                    rewrite( output_file );

                    {
                      The regional compilation may have too many sites to fit on one sheet of paper so the
                      compilation continues on additional sheets.  The maximum number of sites printed
                      on one sheet of paper equals the value of the variable "sites_per_regional_report_page".
                    }
                    sites_per_regional_report_page := trunc( ( total_columns - length( blank_dates_line ) ) /
                                                            ( 1 + width_of_entry_field + width_of_entry_field ) );

                    writeln( output_file, printer_init_sequence );

                    start_address := start_region_directory_address;
                    end_address   := start_address + sites_per_regional_report_page - 1;
                    if   end_address > end_region_directory_address
                    then end_address := end_region_directory_address;

                    repeat

                         writeln( output_file );
                         writeln( output_file, ' Regional Report of Lifestage Development' );
                         writeln( output_file, ' ========================================' );

                         region_report_header( region_name );
                         regional_development_report( start_address, end_address );
                         region_report_trailer;
```

```
                            {
                              If necessary, get the addresses of the next set of sites.
                            }
                            start_address := end_address + 1;
                            end_address   := end_address + sites_per_regional_report_page;
                            if   end_address > end_region_directory_address
                            then end_address := end_region_directory_address;

                       until ( start_address > end_region_directory_address );

                       first_entry := next;

                       writeln( output_file, printer_term_sequence );

                       close( output_file ); { no function on logical devices }

                   end; { while first_entry }

    end; { procedure print_list_of_regional_reports }


begin

    writeln;
    writeln( ' Task : print Regional reports' );
    writeln;

    mark( top_of_heap );
    first_entry := nil;

    get_yes_no_reply( 'a report for all the regions', response );
    if   response[ 1 ] = 'Y'
    then construct_list_of_all_regions
    else construct_list_of_requests;

    if   first_entry <> nil
    then
         begin
             { If the list of requested regions is not empty,
               then proceed with constructing the report. }
             make_line_prefixes;
             print_list_of_regional_reports;
         end; { if first_entry }

    release( top_of_heap );

end; { procedure print_regional_reports }
```

```
GRAFTRAP.PAS
------------

procedure graph_trap_catch;
    {
      This procedure plots a set of graphs for each requested site.
      There is one graph for each pest which is trapped at the site.
    }

    { The following Const, Type and Var declarations are needed to execute a graphics screen dump to the printer. }
    const
        IntNumber = 5;   { Interrupt number for graphics screen dump }
    type
        RegPack = record
                      AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer;
                  end;
    var
        Result                  : RegPack;
    { end graphics screen dump declarations }


    { screen plot parameters }
    const
        dots_per_column =  8; { 8 high resolution dots per text mode column }
        dots_per_row    =  8; { 8 high resolution dots per text mode row    }
        rows_in_graph   = 20; { use 20 text rows for the graph height }
        row_increment   =  2; { label every other row; must be a divisor of "rows_in_graph" }

        { text mode locations of the site, region, season, today and pest fields }
        site_row   = 1; site_column   =  1;
        region_row = 2; region_column =  1;
        season_row = 1; season_column = 60;
        today_row  = 2; today_column  = 60;
        pest_row   = 1; pest_column   = 20;

        { ( x,y ) coordinates of the origin and the opposite corner of the
          plot "box" measured in high resolution dot coordinates }
        origin_x =  88;
        origin_y = 179;
        corner_x = 631;
        corner_y =  19; { corner_y = origin_y - ( dots_per_row * rows_in_graph ) }

        number_of_x_ticks = 11;
    { end screen plot parameters }

    type
        output_device_type = ( no_device, screen, printer );

        want_graph_type = array[ first_pest..last_pest ] of boolean;

        {
          Print requests are stored in a dynamic variable because the number
      of requests is dynamic.  Also, dynamic variables are stored in the
      heap instead of the data segment.  This allows the list of requests
      to grow quite large and reduces the demand on the data segment.

      The number of requests can exceed the number of sites because the
      user may place several requests for a specific printout, i.e. copies.
    }
    graph_request_pointer = ^graph_request_entry;
    graph_request_entry   = record
                                 directory_address : integer;
                                 want_graph        : want_graph_type;
                                 output_device     : output_device_type;
                                 next              : graph_request_pointer;
                            end; { record }

var
    top_of_heap : ^integer;
    first_entry, last_entry, new_entry : graph_request_pointer;

    response            : line;
    output_file         : text;
    output_logical_file : output_device_type;


function do_yes_no_reply( prompt : line; var OK : boolean ) : line;
    var
        ans : line;

    begin

        OK := true;

        write( ' Do you want ' + prompt + ' ? ' );
        readln( ans );
        ans := upper_case( delete_blanks( ans ) );
        if   ans = ''    { null answer not allowed }
        then ans := ' ';
        if   not ( ans[ 1 ] in [ 'Y', 'N' ] )
        then
             begin
                 OK := false;
                 writeln( ' Enter either YES or NO.' );
             end; { if not ( ans[ 1 ] in [ 'Y', 'N' ] ) }

        do_yes_no_reply := ans;

    end; { function do_yes_no_reply }


procedure get_yes_no_reply( prompt : line; var answer : line );
    var
        ans : line;
        OK  : boolean;
    begin

        repeat
            ans := do_yes_no_reply( prompt, OK );
        until OK;

        answer := ans;

    end; { procedure get_yes_no_reply }


procedure fetch_wanted_graphs( var want_graph : want_graph_type );
    {
      This procedure prompts the user for which graphs are desired and
      stores the answers as boolean values.
    }
    var
        prompt, ans : line;
        pest : kinds_of_sites;
    begin

        for pest := succ( first_pest ) to pred( last_pest ) do
            if   pest_trapped_at_some_site[ pest ]
            then
                 begin
                     prompt := 'a graph for ' + pest_name_vector[ pest ];
                     get_yes_no_reply( prompt, ans );
                     want_graph[ pest ] := ( ans[ 1 ] = 'Y' );
                 end; { if pest_trapped_at_some_site[ pest ] }

    end; { procedure fetch_wanted_graphs }


procedure construct_plot_list_of_all_sites;
    {
      This procedure constructs a list consisting of all of the sites in the
      MASTER directory.  The user is asked which graphs she wants to plot.  The same
      set of graphs is plotted for each site.  This reporting option allows only one
      hardcopy printout.
    }
    var
        site_name               : disk_file_name;
        full_site_name          : full_site_name_type;
        site_exists             : boolean;
        site_directory_address  : integer;
        want_graph              : want_graph_type;
        pest                    : kinds_of_sites;

    begin

        fetch_wanted_graphs( want_graph );
        { start at the beginning of the list of sites }
        site_directory_address := 0;
        increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        repeat

              new( new_entry ); { allocate new variable }

              new_entry^.directory_address := site_directory_address;
              for pest := succ( first_pest ) to pred( last_pest ) do
                  new_entry^.want_graph[ pest ] := want_graph[ pest ];
              new_entry^.output_device := printer;

              if   first_entry = nil
              then first_entry := new_entry
              else last_entry^.next := new_entry;

              last_entry := new_entry;
              last_entry^.next := nil;

              increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

        until ( site_directory_address = 1 ); { stop when back to the begining }

    end; { procedure construct_plot_list_of_all_sites }


procedure vertical_tick_mark( x_dot, y_dot, height : integer );
    {
      A procedure to plot a vertical line the top of which is located
      at screen position ( x_dot, y_dot ) and which has a height of
      "height" dots.
    }
    var
        y_end : integer;
    begin

        y_end := y_dot + height; { must add because screen coordinates go from top to bottom }
        draw( x_dot, y_dot, x_dot, y_end, screen_color_code );

    end; { procedure vertical_tick_mark }


procedure plot_single_graph( directory_address : integer; pest : kinds_of_sites; output_device : output_device_type );
    {
      This procedure plots a single trap catch graph of the pest indicated
      from the site indicated.  The graph is sent to the device "output_device".
      If output_device = no_device, no plot graph is produced.
    }
    type
        trap_data_record = record
```

```
                              { indicates kind of data : N - numeric, M - missing, B - blank ( not calculated ) }
                              status : char;

                              day    : integer; { Julian day }
                              mean   : real;    { mean count }
                         end; { record }
var
    site_name                     : disk_file_name;
    region                        : region_name_type;
    season, today, pest_name      : line;
    min_x, max_x                  : integer;
    min_y, max_y, x_scale, y_scale,
       delta_value, max_y_on_axis : real;
    trap_record_number            : integer;
    trap_record                   : array[ 1..366 ] of trap_data_record;


procedure calculate_dot_position( x : integer; y : real; var x_dot, y_dot : integer );
    {
      A procedure to convert an ( x, y ) coordinate pair to a ( x_dot, y_dot )
      screen coordinate pair.  The origins and axes scales in both
      coordinate systems are passed as variables global to this procedure. The
      coordinates of the origin in the ( x, y ) coordinate system, ( min_x, min_y ),
      are passed as global variables, also.
    }
    begin
        x_dot := origin_x + round( ( x - min_x ) * x_scale );
        y_dot := origin_y + round( ( y - min_y ) * y_scale );
    end; { procedure calculate_dot_position }


procedure get_trap_data( site_name : disk_file_name; pest : kinds_of_sites;
                         var min_y, max_y, delta_value, max_y_on_axis : real );
    {
      This procedure reads the trap catch file and records the mean counts.  The data
      are passed to the variables "trap_record_number" and "trap_record" which are
      global to this procedure.

      This procedure also calculates the maximum y-axis value for the graph.
    }
    const
        missing_string = 'MISSING';
    var
        trap_file      : text;
        pest_file_name : line;
        condition_code : integer;
        status         : char;
        date_str       : string[ width_of_date_field ];
        average_str    : string[ width_of_average_field ];
        dummy, mean    : real;

    begin
```

```
min_y :=  0.0; { always start y-axis at 0.0 }
max_y := -1.0; { an actual value must be >= 0.0 }

pest_file_name := PMS_root_directory + Year_path + site_name + '\' + pest_file_name_vector[ pest ] + '.TRP';
assign( trap_file, pest_file_name );
reset( trap_file );

{ The first record is used to calculate the maximum number of traps
  set out for this pest.  It does not contain any actual trap count data,
  hence it is ignored in this procedure. }
readln( trap_file );

trap_record_number := 0;
while not eof( trap_file ) do
    begin

        trap_record_number := trap_record_number + 1;

        readln( trap_file, date_str, average_str );

        average_str := upper_case( delete_blanks( average_str ) );
        val( average_str, dummy, condition_code );

        { Actual means are >= 0.0 since trap counts >= 0.  Thus, the program
          may use a negative mean to indicate an exceptional condition. }

        status := 'X';    { unknown exception condition }
        mean   := -100.0;

        if   average_str = missing_string
        then
             begin
                 status := 'M'; { Missing }
                 mean   := -1.0;
             end;

        if   average_str = ''
        then
             begin
                 status := 'B'; { Blank }
                 mean   := -2.0;
             end;

        if   condition_code = 0
        then
             begin
                 status := 'N'; { Normal }
                 mean   := dummy;
                 if   mean > max_y
                 then max_y := mean;
             end; { if condition_code = 0  }
```

```
            trap_record[ trap_record_number ].status := status;
            trap_record[ trap_record_number ].day    := numeric_date_to_Julian( date_str );
            trap_record[ trap_record_number ].mean   := mean;

        end; { while }

    close( trap_file );

    { calculate maximum value on the y-axis }

    delta_value := row_increment * ( ( max_y - min_y ) / rows_in_graph );

    { recalculate so that y-axis tick marks are at integer valued intervals }
    if   delta_value > trunc( delta_value )
    then delta_value := trunc( delta_value ) + 1;

    max_y_on_axis := ( delta_value * rows_in_graph ) / row_increment;

  end; { procedure get_trap_data }


procedure blank_graph( site_name : disk_file_name; region : region_name_type;
                       pest_name, season, today : line;
                       min_x, max_x : integer; min_y, max_y, delta_value, max_y_on_axis : real );
  {
    This procedure plots a graph with identification fields, but without any
    trap count data points.
  }
  const
      tick_offset = 2;
  type
      two_chars = string[ 2 ];
  var
      text_row, text_col, row0, col0,
          x_dot, y_dot, height                : integer;
      value                                   : real;

      MMDD_first, MMDD_last, MMDD             : numeric_date_string_type;
      MM_first, DD_first, MM_last, DD_last,
          MM, DD                              : two_chars;
      month_first, month_last, month,
          day_first, day_last, day,
          x_Julian, error_code                : integer;

  procedure write_month_abbreviation( MM, DD : two_chars );
      {
        This procedure writes the month's 3 letter abbreviation below the x-axis.
        The abbreviation is written beneath the day of the month given by DD.
      }
      var
          MMDD                   : numeric_date_string_type;
        Month_abbreviation    : string[ 3 ];
        Month_day             : line;
        row0, col0, x_Julian : integer;

    begin

        MMDD := MM + DD;
        Month_day := date_string_from_numeric( MMDD );

        x_Julian := numeric_date_to_Julian( MMDD );
        calculate_dot_position( x_Julian, min_y, x_dot, y_dot );
        col0 := round( x_dot / dots_per_column ) - 1;
        row0 := tick_offset + 1 + trunc( ( origin_y / dots_per_row ) );
        GotoXY( col0, row0 );

        Month_abbreviation := Month_day[ 1 ] + Month_day[ 2 ] + Month_day[ 3 ];
        write( Month_abbreviation );

    end; { procedure write_month_abbreviation }


procedure write_x_axis_tick_mark( MM : two_chars; var day : integer );
    {
      This procedure plots tick marks on the x-axis.  Double height tick marks
      indicate the first of the month.  Single height ticks indicate other dates.
    }
    begin

        str( day : 2, DD );
        DD := delete_blanks( DD );
        if   ( length( DD ) < 2 )
        then DD := '0' + DD;
        MMDD := MM + DD;

        x_Julian := numeric_date_to_Julian( MMDD );
        calculate_dot_position( x_Julian, min_y, x_dot, y_dot );
        if   day > 01
        then height :=  5
        else height := 10;
        vertical_tick_mark( x_dot, y_dot, height );

        day := day + 7; { tick mark every 7 days }

    end; { procedure write_x_axis_tick_mark }

begin

    { identification lines }
    GotoXY( site_column, site_row );
    write( 'SITE   : ' + site_name );
    GotoXY( region_column, region_row );
    write( 'REGION : ' + region );
```

```
GotoXY( season_column, season_row );
write( 'SEASON : ' + season );
GotoXY( today_column, today_row );
write( 'TODAY  : ' + today );
GotoXY( pest_column, pest_row );
write( 'PEST  : ' + pest_name );

{ plot x- and y-axes }
draw( origin_x, corner_y, origin_x, origin_y, screen_color_code );
draw( origin_x, origin_y, corner_x, origin_y, screen_color_code );
{ rest of box }
draw( corner_x, origin_y, corner_x, corner_y, screen_color_code );
draw( corner_x, corner_y, origin_x, corner_y, screen_color_code );

{ y-axis label }
GotoXY( 1, 10 );
write( 'Count' );
GotoXY( 1, 11 );
write( ' per' );
GotoXY( 1, 12 );
write( 'Trap' );

{ ***  plot y-axis values and ticks marks   *** }

value    := max_y_on_axis;
text_row := 3;
repeat
     GotoXY( 7, text_row );
     write( value : 4 : 0, '-' );
     value := value - delta_value;
     text_row := text_row + row_increment;
until ( value < 0.0 );

{ ***  plot x-axis values and tick marks  *** }

{ first month and day on the x-axis }
MMDD_first := Julian_to_numeric_date( min_x );
MM_first   := MMDD_first[ 1 ] + MMDD_first[ 2 ];
DD_first   := MMDD_first[ 3 ] + MMDD_first[ 4 ];
val( MM_first, month_first, error_code );
val( DD_first, day_first,   error_code );

{ last month and day on the x-axis }
MMDD_last  := Julian_to_numeric_date( max_x );
MM_last    := MMDD_last[ 1 ] + MMDD_last[ 2 ];
DD_last    := MMDD_last[ 3 ] + MMDD_last[ 4 ];
val( MM_last, month_last, error_code );
val( DD_last, day_last,   error_code );

{ date on the left end of the x-axis }
col0 := round( origin_x / dots_per_column );
row0 := tick_offset + trunc( ( origin_y / dots_per_row ) );
```

```
GotoXY( col0, row0 );
write( DD_first );
MMDD := MMDD_first;
x_Julian := numeric_date_to_Julian( MMDD );
calculate_dot_position( x_Julian, min_y, x_dot, y_dot );
vertical_tick_mark( x_dot, y_dot, 3 );

{ tick marks in the first month }
{
  The x-axis may begin part way into the month.  Therefore, the procedure must search for
  the first tick marked day after the day which begins the x-axis.
}
day := 01;
repeat
     day := day + 7;
until ( day > day_first );

repeat
     write_x_axis_tick_mark( MM_first, day );
until ( day > 29 );

{ position the month abbreviation within the month interval }
if   ( ( max_x - min_x < 62 ) or ( DD_first < '15' ) )
then
     begin

         if   DD_first < '15'
         then DD := '15'
         else str( day_first + 1 : 2, DD );

         write_month_abbreviation( MM_first, DD );

     end; { if ( ( max_x - min_x < 62 ) or ( DD_first < '15' ) ) }

{ tick marks for the remaining months }
for month := month_first + 1 to month_last - 1 do
    begin
        str( month : 2, MM );

        day := 01;
        repeat
             write_x_axis_tick_mark( MM, day );
        until ( day > 29 );

        { center the month abbreviation within the month interval }
        write_month_abbreviation( MM, '15' );

    end; { for month }

{ date on the right end of the x-axis }
col0 := round( corner_x / dots_per_column );
row0 := 2 + trunc( ( origin_y / dots_per_row ) );
```

```
        GotoXY( col0, row0 );
        write( DD_last );
        MMDD := MMDD_last;
        x_Julian := numeric_date_to_Julian( MMDD );
        calculate_dot_position( x_Julian, min_y, x_dot, y_dot );
        vertical_tick_mark( x_dot, y_dot, 3 );

        { tick marks in the last month }
        if   month_last > month_first
        then
             begin

                 day := 01;
                 while  ( day < day_last ) do
                        write_x_axis_tick_mark( MM_last, day );

                 { position the month abbreviation within the month interval }
                 if   ( ( max_x - min_x < 62 ) or ( DD_last > '15' ) )
                 then
                      begin

                          if   DD_last > '15'
                          then DD := '15'
                          else str( day_last - 1 : 2, DD );

                          write_month_abbreviation( MM_last, DD );

                      end; { if ( ( max_x - min_x < 62 ) or ( DD_last > '15' ) ) }

             end; { if month_last > month_first }

    end; { procedure blank_graph }

procedure trap_count_plot;
    {
      This procedure plots the line graph of the trap count data.
    }
    var
        i, j, day, day0, day1 : integer;
        mean, mean0, mean1    : real;
        x_dot_0, y_dot_0,
        x_dot_1, y_dot_1      : integer;
        status                : char;
        normal_line           : boolean;

    procedure draw_heavy_line( x_dot_0, y_dot_0, x_dot_1, y_dot_1, screen_color_code : integer );
        {
          This procedure draws a heavy line by drawing two lines slightly displaced from one another.
        }
        const
            threshold = 0.416; { = 200 y-pixels / 480 x-pixels;
```

```
                         use 480 since screen horiz:vert aspect ratio is 4:3
                         and total horizontal pixels = 640

                         This threshold is equivalent to a 45 degree slope.
                       }
    var
      slope            : real;
      delta_x, delta_y : integer;
    begin

      if   x_dot_1 - x_dot_0 > 0
      then slope := abs( ( y_dot_1 - y_dot_0 ) / ( x_dot_1 - x_dot_0 ) )
      else slope := threshold + 1.0;

      { Displace the two lines in such a way as to maximize the visual distance between them. }
      if   slope < threshold
      then
         {
           If the line has a slope < 45 degrees,
           then displace the lines vertically.
         }
         begin
           delta_x := 0;
           delta_y := 1;
         end
      else
         {
           If the line has a slope >= 45 degrees,
           then displace the lines horizontally.
         }
         begin
           delta_x := 2;
           delta_y := 0;
         end;
      { if slope < threshold }

      draw( x_dot_0 + delta_x, y_dot_0 + delta_y, x_dot_1 + delta_x, y_dot_1 + delta_y, screen_color_code );
      draw( x_dot_0 - delta_x, y_dot_0 - delta_y, x_dot_1 - delta_x, y_dot_1 - delta_y, screen_color_code );

    end; { procedure draw_heavy_line }


  begin

    if   trap_record_number > 0
    then
      begin
         {
           Note : Trap count data are stored in reverse date order.
                  Therefore, in order to plot the graph from earliest date to
                  latest date, the records must be processed in descending order.
         }
```

```
{ get the first valid data point }
i := trap_record_number + 1;
repeat
      i := i - 1;
      day  := trap_record[ i ].day;
      mean := trap_record[ i ].mean;
until ( ( i = 1 ) or ( ( day > min_x ) and ( trap_record[ i ].status = 'N' ) ) );

if   ( ( i > 0 ) and ( trap_record[ i ].status = 'N' ) )
then
     begin
        {
          If there exists at least one valid point, then render graph.
        }

        normal_line := true; { single width line for graph }

        if   i < trap_record_number
        then
             { suppress any graph line to the left of the left edge of the graph }
             begin
                  { previous point }
                  day0  := trap_record[ i + 1 ].day;
                  mean0 := trap_record[ i + 1 ].mean;

                  { interpolate at the left edge of the graph }
                  mean := mean0 + ( min_x - day0 ) * ( ( mean - mean0 ) / ( day - day0 ) );
                  day  := min_x;

             end; { if i < trap_record_number }

        { calculate point on left edge of graph }
        calculate_dot_position( day, mean, x_dot_0, y_dot_0 );

        { get point immediately to the right of the left edge of the graph }
        day  := trap_record[ i ].day;
        mean := trap_record[ i ].mean;

        repeat
             calculate_dot_position( day, mean, x_dot_1, y_dot_1 );

             if   normal_line
             then draw( x_dot_0, y_dot_0, x_dot_1, y_dot_1, screen_color_code )
             else
                  begin
                       draw_heavy_line( x_dot_0, y_dot_0, x_dot_1, y_dot_1, screen_color_code );
                       normal_line := true; { assume next line segment will be single width }
                  end; { if normal_line }

             x_dot_0 := x_dot_1;
             y_dot_0 := y_dot_1;
     { get next point }
     repeat
          i      := i - 1;
          status := trap_record[ i ].status;
          if   status = 'M'
          then normal_line := false;
     until ( ( i = 1 ) or ( status = 'N' ) );

     day    := trap_record[ i ].day;
     mean   := trap_record[ i ].mean;

until ( ( i = 1 ) or ( day > max_x ) or ( not ( trap_record[ i ].status in [ 'M', 'N' ] ) ) );

case trap_record[ i ].status of

     'B' : begin
                { For this version, do nothing }
           end; { case 'B' }

     'N' : { plot last point }
           begin

               if   day > max_x
               then
                    begin
                        { suppress any graph line to the right of the right edge of the graph }
                        { get previous valid point }
                        j := i; { start at current valid point }
                        repeat
                             j := j + 1; { search backward }
                        until ( trap_record[ j ].status = 'N' );

                        day0  := trap_record[ j ].day;
                        mean0 := trap_record[ j ].mean;

                        { interpolate at the right edge of the graph }
                        mean := mean0 + ( max_x - day0 ) * ( ( mean - mean0 ) / ( day - day0 ) );
                        day  := max_x;
                    end; { if day > max_x }

               { calculate y value at right edge of x-axis }
               calculate_dot_position( day, mean, x_dot_1, y_dot_1 );

               if   normal_line
               then draw( x_dot_0, y_dot_0, x_dot_1, y_dot_1, screen_color_code )
               else
                    begin
                        draw_heavy_line( x_dot_0, y_dot_0, x_dot_1, y_dot_1,
                                         screen_color_code );
                        normal_line := true;
                    end; { if normal_line }
```

```
                                      end; { case 'N' }

                                'M' : begin
                                          { For this version, do nothing }
                                      end; { case 'B' }

                                'X' : begin
                                          { For this version, do nothing }
                                      end; { case 'B' }

                                else  begin
                                          { For this version, do nothing }
                                      end; { case else }

                           end; { case trap_record[ 1 ].status }
                      end
                 else
                      begin
                           GotoXY( 30, 12 );
                           write( 'ALL DATA ARE MISSING.' );
                      end;
                 { end if ( ( 1 > 0 ) and ( trap_record[ 1 ].status = 'N' ) ) }
             end
        else
             begin
                 GotoXY( 30, 12 );
                 write( 'NO TRAP DATA TO DATE.' );
             end;
        { end if trap_record_number > 0 }

    end; { procedure trap_count_plot }


{ main block of procedure plot_single_graph }
begin

    if   output_device <> no_device
    then
         begin

             site_name := master_directory[ directory_address].site_name;
             region    := master_directory[ directory_address].region;
             pest_name := pest_name_vector[ pest ];
             today     := date_string_from_Julian( todays_Julian_date );
             min_x     := trap_plot_x_axis[ pest, 1 ];
             max_x     := trap_plot_x_axis[ pest, 2 ];

             get_trap_data( site_name, pest, min_y, max_y, delta_value, max_y_on_axis );

             x_scale   := ( corner_x - origin_x ) / ( max_x - min_x );
             y_scale   := ( corner_y - origin_y ) / ( max_y_on_axis - min_y );
```

```
            blank_graph( site_name, region, pest_name, Year_subdirectory, today,
                         min_x, max_x, min_y, max_y, delta_value, max_y_on_axis );
            trap_count_plot;

            if   output_device = printer
            then
                 begin
                     Intr( IntNumber, Result );
                     write( LST, form_feed );
                 end; { if output_device = printer }

        end; { if output_device <> no_device }

    end; { procedure plot_single_graph }


procedure construct_plot_list_of_requests;
    {
      This procedure first prompts the user for a site name and then displays a graph
      for each trapped pest on the screen.  The procedure asks the user if she would
      like a printer copy and stores her answer as a boolean value.  Site names may be
      entered in any order.  This option allows multiple requests for the same graph,
      i.e. copies, by allowing the user to request a site more than once.
    }

    var
        ans, response, prompt           : line;
        site_name                       : disk_file_name;
        full_site_name                  : full_site_name_type;
        site_exists                     : boolean;
        site_directory_address          : integer;
        want_graph                      : want_graph_type;
        pest                            : kinds_of_sites;
        pests_trapped, OK_trap_site, OK : boolean;
        ch                              : char;

    begin

        site_directory_address := 1;
        site_name := master_directory[ site_directory_address].site_name;

        repeat

             { make sure the site gathers pest trap data }
             next_trap_site( site_name, site_directory_address );

             repeat

                  fetch_site_name( '', response, site_name, full_site_name, site_exists, site_directory_address );
                  if   response <> nomore
                  then
```

```
          begin

              OK_trap_site := valid_trap_site( site_name, site_directory_address );
              if   not OK_trap_site
              then
                   begin
                       writeln( beep + ' Requested site does not trap any pests.' );
                       next_trap_site( site_name, site_directory_address );
                   end; { if not OK_trap_site }

          end; { if response <> nomore }

until ( OK_trap_site or ( response = nomore ) );

if   response <> nomore
then
     begin
         {
           Note : There is a potentially inefficient use of memory in this block.
                  If the user does not select any graphs for printing, the block
                  really does not need to create an entry for it; however, the current
                  code creates an entry anyway.
         }

         new( new_entry ); { allocate new variable }

         new_entry^.directory_address := site_directory_address;

         for pest := succ( first_pest ) to pred( last_pest ) do
             if   master_directory[ site_directory_address ].trapped[ pest ]
             then
                  begin

                      HiRes;
                      HiResColor( screen_color_code );

                      plot_single_graph( site_directory_address, pest, screen );

                      repeat
                           GotoXY( 1, 25 );
                           write( ' Do you want a printer copy ? ' );
                           read( Kbd, ch );
                           OK := ch in [ 'Y', 'y', 'N', 'n' ];
                           if   not OK
                           then write( '   Enter either Y or N.' );
                      until OK;

                      new_entry^.want_graph[ pest ] := ( ch in [ 'Y', 'y' ] );

                      TextMode( BW80 );

                  end; { if master_directory[ site_directory_address ].trapped[ pest ] }
```

```
                       new_entry^.output_device := printer;

                       if   first_entry = nil
                       then first_entry := new_entry
                       else last_entry^.next := new_entry;

                       last_entry := new_entry;
                       last_entry^.next := nil;

                       increment_to_next_site( site_name, full_site_name, site_exists, site_directory_address );

                   end; { if response }

        until ( response = nomore );

    end; { procedure construct_plot_list_of_requests }


procedure plot_list_of_graphs;
    {
      This procedure reads the list of requested graphs and plots them
      on the printer.
    }
    var
        site_name, nearest_weather_site   : disk_file_name;
        full_site_name                    : full_site_name_type;
        region                            : region_name_type;
        pest                              : kinds_of_sites;

    begin

        while first_entry <> nil do
              with first_entry^ do
                   begin

                        full_site_name       := master_directory[ directory_address ].full_name;
                        site_name            := master_directory[ directory_address ].site_name;
                        nearest_weather_site := master_directory[ directory_address ].nearest_weather_site;
                        region               := master_directory[ directory_address ].region;

                        for pest := succ( first_pest ) to pred( last_pest ) do
                            if   ( master_directory[ directory_address ].trapped[ pest ] and want_graph[ pest ] )
                            then
                                 begin
                                     HiRes;
                                     HiResColor( screen_color_code );
                                     plot_single_graph( directory_address, pest, output_device );
                                     TextMode( BW80 );
                                 end; { if }

                        first_entry := next;
```

```
                  end; { while first_entry }

    end; { procedure plot_list_of_graphs }


begin

    writeln;
    writeln( ' Task : plot trap catch Graphs' );
    writeln;

    mark( top_of_heap );
    first_entry := nil;

    get_yes_no_reply( 'all graphs for all the sites', response );
    if   response[ 1 ] = 'Y'
    then construct_plot_list_of_all_sites
    else construct_plot_list_of_requests;

    plot_list_of_graphs;

    release( top_of_heap );

end; { procedure graph_trap_catch }
```